Pentium® Pro Processor Specification Update

Release Date: October 1998

Order Number: 242689-032

The Pentium® Pro processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are documented in this Specification Update.

CONTENTS

REVISION HISTORY

Date of Revision	Version	Description
November 1995	-001	This document is the first Specification Update for the Intel Pentium® Pro processor.
December 1995	-002	Added Errata 29 through 38, and 3AP. Included additional S-specs. Added a case to Erratum 3. Clarified Errata 2, 6, 8, 10, 11, and 26.
January 1996	-003	Added Errata 39 through 42. Updated the status for Erratum 10. Added documentation for 166-, 180-, and 200-MHz processors.
February 1996	-004	Clarified Errata 11, 22, 30, and 34. Updated status for Errata 10, 15, 40, and 42. Added CPUID information for determining L2 cache size. Added S-specs and errata column for sA1 stepping.
March 1996	-005	Added Errata 43 through 46. Added Specification Change 1 and Documentation Change 1. Clarified Erratum 2 and corrected Errata 7 and 39. Added L2 cache stepping information.
April 1996	-006	Added Erratum 47. Updated General Information section with new S-specs and marking diagrams.
May 1996	-007	Corrected Erratum 43. Added Erratum 48.
June 1996	-008	Added Specification Change 2, Errata 49 and 4AP, Specification Clarifications 1 through 3, and Documentation Changes 2 and 3. Updated status of Errata 7, 33, and 43. Updated S-spec table with additional and corrected sA0 and sA1 S-spec information.
July 1996	-009	Added Specification Change 3, Errata 50 and 5AP, Specification Clarifications 4 and 5, and Documentation Change 4. Updated Erratum 43.
August 1996	-010	Updated Errata 50 and 2AP. Added Specification Changes 4 through 6, Errata 51 and 6AP, and Specification Clarifications 6 and 7.
September 1996	-011	Updated Specification Change 1. Added Errata 52 through 54. Added Specification Clarifications 8 and 9.
October 1996	-012	Updated Errata 49 and 54. Added Specification Clarifications 10 through 12. Added Documentation Changes 5 through 8.
November 1996	-013	Updated S-spec table. Updated Errata 33, 49, 51, 54, 5AP and 6AP. Added Errata 55, 7AP and 8AP, Specification Clarifications 13 and 14, and Documentation Changes 9 through 11. Updated Specification Change 1.
December 1996	-014	Updated S-spec table and corrected the L2 cache stepping for SY040. Updated marking diagrams for sB1 components. Updated Specification Change 1, Erratum 54. Added Specification Change 7 and Errata 56, 57, and 9AP.
January 1997	-015	Added Errata 58 and 59. Added Specification Clarification 15.
March 1997	-017	Added Documentation Change 13. Updated S-Spec table.

Date of Revision	Version	Description
April 1997	-018	Added Errata 60 and 61. Added Specification Clarifications 16, 17, 18, and 19. Added Documentation Changes 14 and 15. Updated Erratum 52 title and description. Updated Specification Clarification 14. Updated Documentation Change 13. Added engineering sample markings and identification information.
June 1997	-019	Added Erratum 62. Added Specification Clarification 20.
August 1997	-020	Added Erratum 63. Added Documentation Change 16.
October 1997	-021	Added Boxed Pentium Pro Processor Markings; Errata 64, 65, 66, 67; Specification Clarification 21; and Documentation Changes 17, 18, 19, 20 and 21. Updated Erratum 46 and 63; Specification Change 1, Specification Clarification 14 and S-Spec table.
November 1997	-022	Updated Erratum 64. Added Specification Clarification 22, 23, and 24.
December 1997	-023	Corrected S-spec table. Updated Erratum 61. Added Errata 68 and 69.
January 1998	-024	Updated S-spec table. Updated Erratum 66. Added Erratum 70. Updated Specification Clarification 14. Added Documentation Changes 22 and 23.
February 1998	-025	Added Documentation Change 24.
March 1998	-026	Updated Errata 31 and 42. Added Erratum 71. Added Specification Clarification 25. Added Documentation Change 25.
April 1998	-027	Updated identification information. Added new markings. Updated S-spec table. Updated Specification Changes 1, 2, 4. Corrected Erratum 2. Added Errata 72, 73, and 74. Updated Specification Clarification 9. Corrected Documentation Change 6. Added Documentation Change 26.
May 1998	-028	Updated Errata 3 and 74. Added Errata 75, 76, and 77. Added Specification Changes 9, and 10. Added Documentation Changes 27, and 28.
June 1998	-029	Added Erratum 78. Updated Specification Clarification 11. Added Specification Clarifications 26 and 27. Added Documentation Changes 29, 30, 31 and 32.
August 1998	-030	Added Errata 79 and 80. Added Documentation Change 33. Updated Erratum 46.
September 1998	-031	Added Erratum 81. Updated Erratum 80.
October 1998	-032	Added Specification Change 11. Updated Errata 2 and 76. Added Errata 82 through 84. Added Specification Clarifications 28 and 29.

PREFACE

This document is an update to the specifications contained in the *Pentium® Pro Family Developer's Manual*, *Volumes 1, 2, and 3* (Order Numbers 242690, 242691, and 242692, respectively) and the *Pentium® Pro Processor with 1 MB L2 Cache at 200 MHz* datasheet (Order Number 243570-001). It is intended for hardware system manufacturers and software developers of applications, operating systems, or tools. It contains Specification Changes, S-Specs, Errata, Specification Clarifications, and Documentation Changes.

Nomenclature

Specification Changes are modifications to the current published specifications. These changes will be incorporated in the next release of the specifications.

S-Specs are exceptions to the published specifications, and apply only to the units assembled under that s-spec.

Specification Clarifications describe a specification in greater detail or further highlight a specification's impact to a complex design situation. These clarifications will be incorporated in the next release of the specifications.

Documentation Changes include typos, errors, or omissions from the current published specifications. These changes will be incorporated in the next release of the specifications.

Errata are design defects or errors. Errata may cause the Pentium® Pro processor's behavior to deviate from published specifications. Hardware and software designed to be used with any given stepping must assume that all errata documented for that stepping are present on all devices.

Identification Information

The Pentium Pro processor can be identified by the following values:

Family[1]	150-, 166-, 180-, and 200-MHz Model 1[2]
0110	0001

NOTES:

1. The Family corresponds to bits [11:8] of the EDX register after RESET, bits [11:8] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the generation field of the Device ID register accessible through Boundary Scan.
2. The Model corresponds to bits [7:4] of the EDX register after RESET, bits [7:4] of the EAX register after the CPUID instruction is executed with a 1 in the EAX register, and the model field of the Device ID register accessible through Boundary Scan.

The Pentium Pro processor's level 2 (L2) cache size can be determined by the following values:

256-Kbyte Unified L2 Cache[1]	512-Kbyte Unified L2 Cache[1]	1-Mbyte Unified L2 Cache[1]
42h	43h	44h

NOTE:

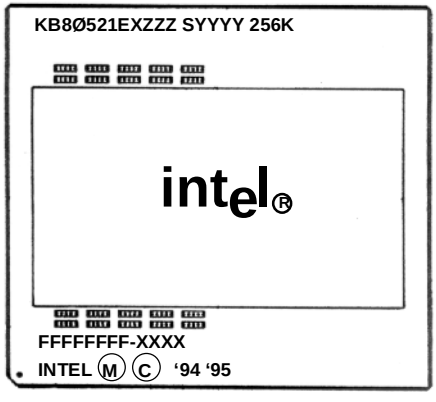
1. For this model of the Pentium® Pro processor, the unified L2 cache size corresponds to the value in bits [3:0] of the EDX register after the CPUID instruction is executed with a 2 in the EAX register. Other Intel microprocessor models or families may move this information to other bit positions or otherwise reformat the result returned by this instruction; generic code should parse the resulting token stream according to the definition of the CPUID instruction.

Specification Update for 150-, 166-, 180-, and 200-MHz Pentium® Pro Processors

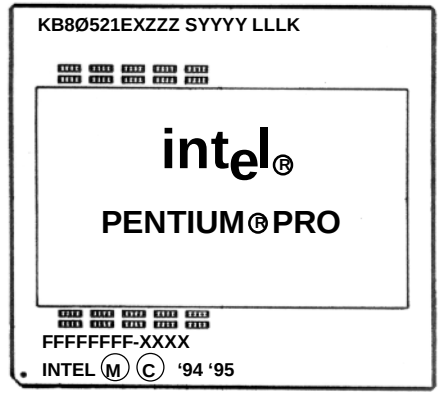
GENERAL INFORMATION

256K and 512K Cache-size Processors - Top Markings

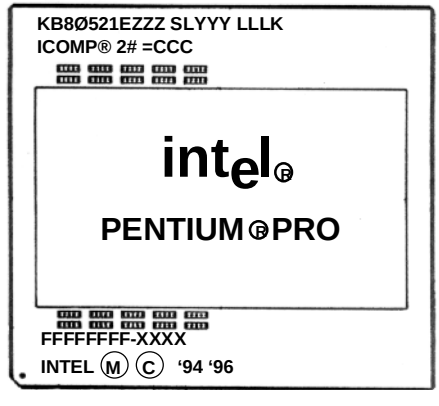
B-Step Production Units – Top:



B-, C-, and sA-Step Production Units – Top:

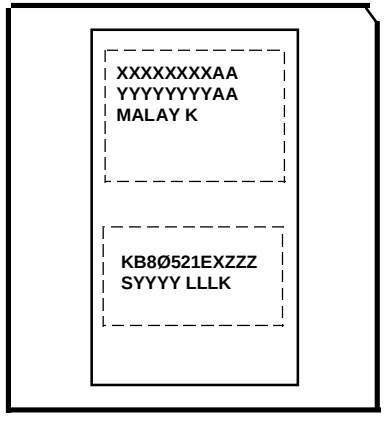


sB1-Step Production Units – Top:

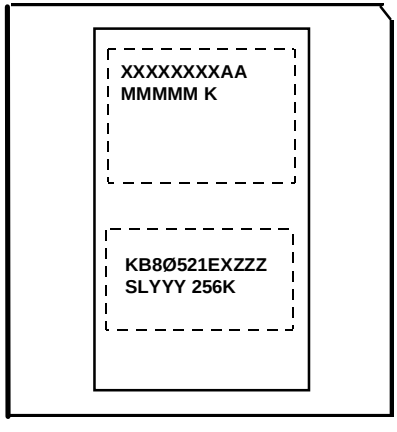


256K and 512K Cache-size Processors - Bottom Markings

B-, C-, and sA-Step Production Units – Bottom:

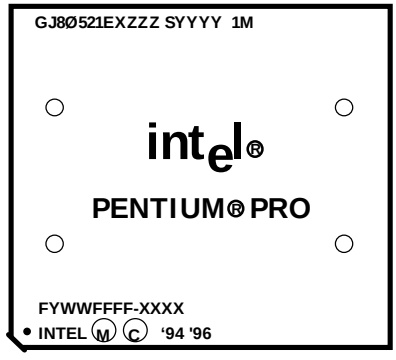


sB1-Step Production Units – Bottom:



1 MB Cache-size Processors Markings

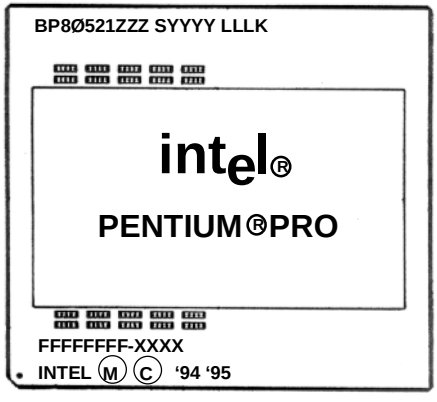
sB1-Step Production Units – Top:



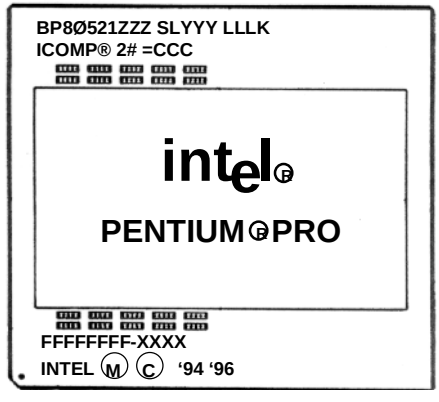
Boxed Pentium® Pro Processor Markings

Top Markings

sA-Step Production Units – Top:

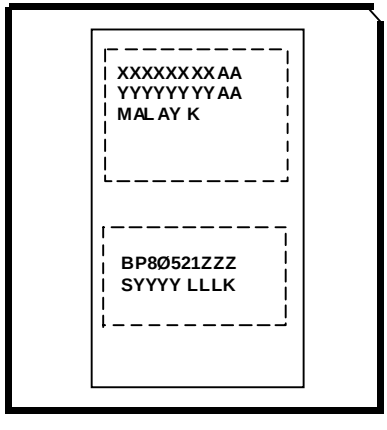


sB1-Step Production Units – Top:

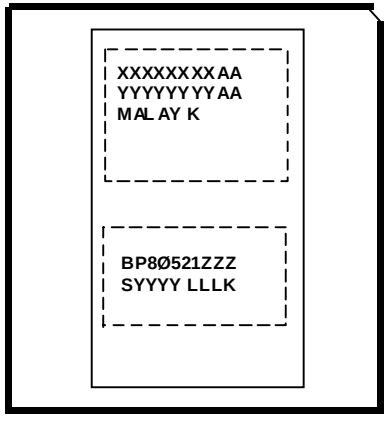


Bottom Markings

sA-Step Production Units – Bottom:



sB1-Step Production Units – Bottom:



NOTES:

- ZZZ = Speed (MHz).
- SYYYY = S-spec Number.
- LLLK = Level 2 Cache Size.
- FFFFFFFF = FPO # (Test Lot Traceability #).
- XXXX = Serialization Code.
- XXXXXXXXAA = Alternative Identification Number.

Top side: inner line indicates boundary of heat spreader.

Basic 150-, 166-, 180-, and 200-MHz Pentium® Pro Processor Identification Information

CPUID										
Type	Family	Model	Stepping	Mfg. Stepping	L2[1] Size/ Cache Stepping	Speed (MHz) Core/Bus	S-Spec	V_{CC}	T_{CASE}	Notes
0	6	1	1	B0	256/α	150/60	SY002	3.1 V ± 5%	0 °C – 85 °C	3
0	6	1	1	B0	256/α	150/60	SY011	3.1 V ± 5%	0 °C – 85 °C	
0	6	1	1	B0	256/α	150/60	SY014	3.1 V ± 5%	0 °C – 85 °C	
0	6	1	2	C0	256/α	150/60	SY010	3.1 V ± 5%	0 °C – 85 °C	
0	6	1	6	sA0[2]	256/α	180/60	SY012	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	6	sA0[2]	256/α	200/66	SY013	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	7	sA1	256/α	200/66	SL245	3.5 V ± 5%	0 °C – 85 °C	7
0	6	1	7	sA1	256/α	200/66	SL247	3.5 V ± 5%	0 °C – 85 °C	7
0	6	1	7	sA1	256/β	180/60	SU103	3.3 V ± 5%	0 °C – 85 °C	8
0	6	1	7	sA1	256/β	200/66	SU104	3.3 V ± 5%	0 °C – 85 °C	8
0	6	1	7	sA1	256/β	180/60	SY031	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	7	sA1	256/β	200/66	SY032	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	7	sA1	512/α	166/66	SY034	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	7	sA1	256/α	180/60	SY039	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	7	sA1	256/β	200/66	SY040	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	7	sA1	512/β	166/66	SY047	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	7	sA1	512/β	200/66	SY048	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	9	sB1	256/β	180/60	SL22S	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	9	sB1	256/β	200/66	SL22T	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	9	sB1	256/β	180/60	SL22U	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	9	sB1	256/β	200/66	SL22V	3.3 V ± 5%	0 °C – 85 °C	9
0	6	1	9	sB1	512/β	166/66	SL22X	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	9	sB1	512/β	200/66	SL22Z	3.3 V ± 5%	0 °C – 85 °C	
0	6	1	9	sB1	256/β	180/60	SL23L	3.3 V ± 5%	0 °C – 85 °C	8
0	6	1	9	sB1	256/β	200/66	SL23M	3.3 V ± 5%	0 °C – 85 °C	8
0	6	1	9	sB1	256/β	200/66	SL254	3.5 V ± 5%	0 °C – 85 °C	7
0	6	1	9	sB1	256/β	200/66	SL255	3.5 V ± 5%	0 °C – 85 °C	7
0	6	1	9	sB1	512/β	166/66	SL2FJ	3.3 V ± 5%	0 °C – 85 °C	8
0	6	1	9	sB1	1024/γ	200/66	SL259	3.3V ± 5%, 3.2 V ± 0.1V	0 °C – 80 °C	10
0	6	1	9	sB1	1024/γ	200/66	SL25A	3.3 V ± 5%, 3.2 V ± 0.1V	0 °C – 80 °C	10

NOTES:

1. **L2 Cache Stepping** refers to the silicon revision of the 256-Kbyte, 512-Kbyte, or 1 Mbyte on-chip L2 cache. The α designation refers to the first production steppings, the β to the second production steppings, and so on.
2. The sA0 stepping is logically equivalent to the C0 stepping, but on a different manufacturing process.
3. The VID pins are not supported on these parts.
4. These are engineering samples only, provided under a Pentium® Pro processor nondisclosure loan agreement.
5. The VID pins are functional but not tested on these parts.
6. These sample parts are equipped with a pre-production 512-Kbyte L2 cache.
7. These components have additional specification changes associated with them:
    a. $V_{CC}P$ = Primary V_{CC} = 3.5 V $\pm$ 5%
    b. P_{MAX} = Max Thermal Design Power = 39.4W @ 200 MHz, 256K L2
    c. $I_{CC}P$ = $V_{CC}P$ Current = 11.9A
    d. The VID pins are not supported on these parts.
    e. T9 = Minimum GTL+ Input Hold Time = 0.9ns
    f. V_{IH}min = Minimum Non-GTL+ Input High Voltage = 2.2 V [Only PICD[1:0] signals affected. Clocks are unaffected]
8. This is a boxed Pentium Pro processor with an unattached fan heatsink.
9. This part also ships as a boxed processor with an unattached fan heatsink.
10. Please reference the *Pentium® Pro Processor with 1 MB L2 Cache at 200 MHz* datasheet, Section 2.13, “DC Specification,” Table 4, “Power and Voltage Specification (Option 1),” and Table 5, “Power and Voltage Specification (Option 2),” for $V_{CC}P$ options.

Summary Table of Changes

The following table indicates the Specification Changes, Errata, Specification Clarifications, or Documentation Changes which apply to the 150-, 166-, 180-, and 200-MHz Pentium Pro processors. Intel intends to fix some of the errata in a future stepping of the component, and to account for the other outstanding issues through documentation or specification changes as noted. This table uses the following notations:

CODES USED IN SUMMARY TABLE

X:	Specification Change, Erratum, Specification Clarification, or Documentation Change applies to the given stepping.
Doc:	Intel intends to update the appropriate documentation in a future revision.
Fix:	Intel is investigating the possibility of fixing this erratum in a future stepping of the component(s).
Fixed:	This erratum is intended to be fixed in a future stepping of the component.
NoFix:	There are no plans to fix this erratum.
(No mark) or (blank box):	This item is fixed in or does not apply to the given stepping.
AP:	APIC related erratum.
Shaded:	This item is either new or modified from the previous version of the document.

NO.	B0	C0	sA0	sA1	sB1	Plans	ERRATA
1	X	X	X	X	X	NoFix	Mixed cacheability of lock variables is problematic in MP systems
2	X	X	X	X	X	NoFix	FP Data Operand Pointer may be incorrectly calculated after FP access which wraps 64-Kbyte boundary in 16-bit code
3	X	X	X	X	X	NoFix	Differences exist in debug exception reporting
4	X	X	X	X	X	NoFix	FLUSH# servicing delayed while waiting for STARTUP_IPI in MP systems
5	X					Fixed	Fast Strings REP MOVS may not transfer all data
6	X					Fixed	Page table base change during task switch using Mode C paging may corrupt EIP
7	X	X	X	X	X	NoFix	Code fetch matching disabled debug register may cause debug exception
8	X	X	X	X	X	NoFix	Mode C paging in SMM causes use of incorrect page tables
9	X					Fixed	Memory indirect near call may corrupt EIP
10	X					Fixed	L2 single-bit correctable error may cancel simultaneous valid data
11	X					Fixed	Page split access before write to CR3 may cause hang
12	X					Fixed	Active A20M# during SMM dump
13	X					Fixed	Split access across 4-Kbyte page boundary may cause hang
14	X	X	X	X	X	NoFix	Checker BIST failure in FRC mode not signaled
15	X	X	X	X	X	NoFix	BINIT# assertion causes FRCERR assertion in FRC mode
16	X	X	X	X		Fixed	Extra page fault may occur on IRET during task switch

NO.	B0	C0	sA0	sA1	sB1	Plans	ERRATA
17	X					Fixed	Some caching models in SMM may cause shutdown
18	X					Fixed	Fast Strings feature re-enabled after INIT event
19	X					Fixed	THERMTRIP# feature not present
20	X	X	X	X		Fixed	OUT instruction, Branch Trace Message may write incorrect data
21		X	X	X		Fixed	THERMTRIP# pin not asserted for catastrophic thermal condition
22	X	X	X	X		Fixed	LBER value may not be updated correctly
23	X	X	X	X	X	NoFix	BTM for SMI will contain incorrect FROM EIP
24	X	X	X	X	X	NoFix	Task switch fault may allow read access of linear address 0h
25	X					Fixed	Low frequencies with 5:2 core to bus clock ratio may fail in FRC
26	X	X	X	X		Fixed	RDPMC cannot be used in conjunction with SMM
27	X	X	X	X		Fixed	PWRGOOD forced to 0 during boundary scan resets TAP
28	X	X	X	X		Fixed	BIST failure not indicated when RUNBIST TAP command used
29	X	X	X	X		Fixed	INVLPG may not invalidate targeted ITLB entry
30	X	X	X	X		Fixed	SMI does not flush TLB entries with PGE enabled
31	X	X	X	X	X	NoFix	I/O restart in SMM may fail after simultaneous MCE
32	X	X	X	X		Fixed	SMBASE reset on INIT# or INIT_IPI
33	X	X	X	X	X	NoFix	MCE due to L2 ECC error gives L1 MCACOD.LL
34	X	X	X	X		Fixed	INVLPG does not invalidate entire 0 to 4-Mbyte region
35	X					Fixed	BINIT# assertion during snoop hit may cause double Machine Check Exception
36	X	X	X	X	X	NoFix	Machine Check Exception handler may not always execute successfully
37	X	X	X	X	X	NoFix	Double ECC error on read may result in BINIT#
38	X	X	X	X		Fixed	RSM cannot return to HALT or SHUTDOWN in 32-bit OS
39	X	X	X			Fixed	Accesses of Modified data may hang system
40	X	X	X	X	X	NoFix	Branch traps do not function if BTMs are also enabled
41	X	X	X			Fixed	Cache line may exist in two different ways in MP system
42	X	X	X	X	X	Fixed	HALT, SHUTDOWN, and STPCLK special cycles not issued
43	X	X	X	X		Fixed	L2 cache may incorrectly report BIST failure
44	X	X	X	X	X	NoFix	LOCK prefix does not generate trap with some instructions
45	X	X	X	X		Fixed	FRCERR may be incorrectly asserted on explicit writeback cycles
46	X	X	X	X	X	NoFix	FP inexact-result exception flag may not be set

NO.	B0	C0	sA0	sA1	sB1	Plans	ERRATA
47	X	X	X	X		Fixed	Snoop hit during RSM segment load may hang system
48	X	X	X	X		Fixed	Mbyte-aligned SMM base address causes SHUTDOWN
49	X	X	X	X	X	NoFix	Machine Check Exception with coexisting large and small pages
50	X	X	X	X		Fixed	WBINVD to some modified addresses hangs system
51	X	X	X	X		Fixed	UC store may be reordered around WC transaction
52	X	X	X	X		Fixed	Spurious IFU Error Reported With MCEs Enabled
53	X	X	X	X	X	NoFix	Extended TRDY# may cause hang with DBSY# asserted
54	X	X	X	X	X	NoFix	Call gates can cause incorrect handling of stack size attribute
55	X	X	X	X	X	NoFix	LBER may be corrupted after some events
56					X	NoFix	SMI during REP string instruction clears CR4
57	X	X	X	X	X	NoFix	BTMs may be corrupted during simultaneous L1 cache line replacement
58	X	X	X	X	X	NoFix	Spurious Machine Check Exception Via IFU Data Parity Error
59	X	X	X	X	X	NoFix	Loss of inclusion in IFU can cause Machine Check Exception
60					X	NoFix	TAP Controller not automatically reset at power-up
61	X	X	X	X	X	NoFix	Erroneous signaling of user mode protection violation
62	X	X	X	X	X	NoFix	Invalid operation not signaled by the FIST instruction on some out of range operands
63	X	X	X	X	X	NoFix	EFLAGS may be incorrect after a multiprocessor TLB shootdown
64	X	X	X	X	X	NoFix	Possible system hang when paging is disabled and re-enabled from uncached memory
65	X	X	X	X	X	NoFix	FLUSH# assertion disables L2 Machine Check Exception reporting
66	X	X	X	X	X	NoFix	Delayed line invalidation issue during 2-way MP data ownership transfer
67	X	X	X	X	X	NoFix	Potential early deassertion of LOCK# during Split-Lock Cycles
68	X	X	X	X	X	NoFix	A20M# may be inverted after returning from SMM and RESET
69	X	X	X	X	X	NoFix	EFLAGS discrepancy on a page fault after a multiprocessor TLB shootdown
70	X	X	X	X	X	NoFix	Near CALL to ESP creates unexpected EIP address
71	X	X	X	X	X	NoFix	Memory type undefined for nonmemory operations
72	X	X	X	X	X	NoFix	Bus protocol conflict with optimized chipsets
73	X	X	X	X	X	NoFix	FP Data Operand Pointer may not be zero after power on or Reset
74	X	X	X	X	X	NoFix	Premature execution of a load operation prior to exception handler invocation

NO.	B0	C0	sA0	sA1	sB1	Plans	ERRATA
75	X	X	X	X	X	NoFix	Read Portion of RMW Instruction may execute twice
76	X	X	X	X	X	NoFix	Intervening writeback may occur during locked transaction
77	X	X	X	X	X	NoFix	MC2_STATUS MSR has Model Specific error code and Machine Check Architecture error code reversed
78	X	X	X	X	X	NoFix	MOV with debug register causes debug exception
79	X	X	X	X	X	NoFix	Data breakpoint exception in a displacement relative near call may corrupt EIP
80	X	X	X	X	X	NoFix	Misprediction in program flow may cause unexpected instruction execution
81	X	X	X	X	X	NoFix	RDMSR and WRMSR to invalid MSR address may not cause GP fault
82	X	X	X	X	X	NoFix	SYSENTER/SYSEXIT instructions can implicitly load “null segment selector” to SS and CS registers
83	X	X	X	X	X	NoFix	PRELOAD followed by EXTEST does not load boundary scan data
84	X	X	X	X	X	NoFix	Far jump to new TSS with D-bit cleared may cause system hang
1AP	X	X	X	X	X	NoFix	APIC access to cacheable memory causes SHUTDOWN
2AP	X	X	X	X	X	NoFix	Possible hang due to catastrophic errors during BSP determination
3AP	X	X	X	X		Fixed	INIT_IPI after STARTUP_IPI-STARTUP_IPI sequence may cause AP to execute at 0h
4AP		X	X	X		Fixed	Small window for system hang after INIT# or INIT_IPI
5AP	X	X	X	X		Fixed	Virtual Wire mode through local APIC may cause Int 15
6AP	X	X	X	X		Fixed	Divide By Zero error on LINT0 glitch
7AP	X	X	X	X	X	NoFix	Write to mask LVT (programmed as EXTINT) will not deassert outstanding interrupt
8AP	X	X	X	X	X	NoFix	APs do not respond to a STARTUP_IPI after an INIT# or INIT_IPI in Low Power Mode
9AP					X	NoFix	Serial bus interrupts may be lost in APIC mixed mode

NO.	B0	C0	sA0	sA1	sB1	Plans	DOCUMENTATION CHANGES
1	X	X	X	X	X	Doc	PE bit number in FPU status word, Volume 2, page 7-50
2	X	X	X	X	X	Doc	GTL+ edge rate range incorrect
3	X	X	X	X	X	Doc	Power-on configuration register documented inconsistently
4	X	X	X	X	X	Doc	GTL+ minimum overshoot specification
5	X	X	X	X	X	Doc	Conditional move opcode incorrect
6	X	X	X	X	X	Doc	32 entries for 4-Kbyte pages in ITLB
7	X	X	X	X	X	Doc	A5# value should be H for Arb Id 1
8	X	X	X	X	X	Doc	Register names incorrect for CPUID return value

NO.	B0	C0	sA0	sA1	sB1	Plans	DOCUMENTATION CHANGES
9	X	X	X	X	X	Doc	Corrections to *Volume 2: Programmer's Reference Manual*
10	X	X	X	X	X	Doc	Remapping WB, WT memory types
11	X	X	X	X	X	Doc	BTM “From” and “To” fields reversed
12	X	X	X	X	X	Doc	Numeric underflow exception description clarification
13	X	X	X	X	X	Doc	Page directory pointer table register present bit identification
14	X	X	X	X	X	Doc	Flushing caches upon entering SMM
15	X	X	X	X	X	Doc	General protection fault description
16	X	X	X	X	X	Doc	Invalid arithmetic operations and masked responses to them relative to FIST/FISTP instruction
17	X	X	X	X	X	Doc	Limit checking and G Flag usage correction
18	X	X	X	X	X	Doc	MCI_ADDR MSR reference section correction
19	X	X	X	X	X	Doc	MC2_STATUS bit description correction
20	X	X	X	X	X	Doc	FCOMI/FCOMIP/FUCOMI/FUCOMIP setting of flags relative to exceptions
21	X	X	X	X	X	Doc	Cache management instructions correction
22	X	X	X	X	X	Doc	PUSH does not pad with zeros
23	X	X	X	X	X	Doc	DR7, bit 10 is reserved
24	X	X	X	X	X	Doc	Additional states that are not automatically saved and restored
25	X	X	X	X	X	Doc	Cache and TLB description correction
26	X	X	X	X	X	Doc	SMRAM State Save Map Contains Documentation Error
27	X	X	X	X	X	Doc	OF and DF of the EFLAGS Register are mislabeled as System Flags
28	X	X	X	X	X	Doc	CS:EIP pushed onto stack prior to code segment limit check
29	X	X	X	X	X	Doc	BREQ# sampled incorrectly
30	X	X	X	X	X	Doc	Corrections to Opcode Maps
31	X	X	X	X	X	Doc	MP initialization protocol algorithm correction
32	X	X	X	X	X	Doc	Interrupt 13-General Protection Exception (#GP)
33	X	X	X	X	X	Doc	Addition to the Lock instruction description

NO.	B0	C0	sA0	sA1	sB1	Plans	SPECIFICATION CLARIFICATIONS
1	X	X	X	X	X	Doc	Signal edge rate for 3.3 V tolerant, APIC, and JTAG signals
2	X	X	X	X	X	Doc	OUTS instruction and subsequent instructions
3	X	X	X	X	X	Doc	Interrupt recognition determines priority
4	X	X	X	X	X	Doc	References to 2-Mbyte pages should include 4-Mbyte pages
5	X	X	X	X	X	Doc	Modification of reserved areas in the SMRAM saved state map
6	X	X	X	X	X	Doc	FRCERR asserted for FRC checker BIST failure

NO.	B0	C0	sA0	sA1	sB1	Plans	SPECIFICATION CLARIFICATIONS
7	X	X	X	X	X	Doc	Deassertion of BREQ*n*# during RESET#
8	X	X	X	X	X	Doc	TLB flush necessary after PDPE change
9	X	X	X	X	X	Doc	LOCK# deasserted between locked sequences
10	X	X	X	X	X	Doc	EXF4# assertion in SMM
11	X	X	X	X	X	Doc	Preventing caching
12	X	X	X	X	X	Doc	Mc*i*_CTL registers not cleared by INIT#
13	X	X	X	X	X	Doc	Stack use during interrupts, exceptions, and CALLs
14	X	X	X	X	X	Doc	Performance-Monitoring counter issues
15	X	X	X	X	X	Doc	Clock jitter measurement clarification
16	X	X	X	X	X	Doc	Exception handler error code bit clarification
17	X	X	X	X	X	Doc	Clock frequencies and ratios and OverDrive® processors
18	X	X	X	X	X	Doc	BERR# pin description clarification
19	X	X	X	X	X	Doc	IERR# pin description clarification
20	X	X	X	X	X	Doc	Propagation of page table entry changes to multiple processors
21	X	X	X	X	X	Doc	APIC bus status cycles interpretation
22	X	X	X	X	X	Doc	Multiple processors protocol and restrictions
23	X	X	X	X	X	Doc	NMI handling while in SMM
24	X	X	X	X	X	Doc	Critical sequence of events during a page fault exception
25	X	X	X	X	X	Doc	POP[ESP] with 16-bit stack size
26	X	X	X	X	X	Doc	Paging must be enabled before enabling the page global enable bit
27	X	X	X	X	X	Doc	SMIACK transaction generated on exiting from SMM
28	X	X	X	X	X	Doc	Software initialization requirements for FRC mode
29	X	X	X	X	X	Doc	Switching to protected mode while in SMM

NO.	B0	C0	sA0	sA1	sB1	Plans	SPECIFICATION CHANGES
1	X	X	X	X	X	Doc	Mixing steppings in MP systems
2	X	X	X	X	X	Doc	Change in PICD[1:0] hold time and valid delay
3	X	X	X	X	X	Doc	Undershoot specification for some GTL+ signals
4						Doc	OverDrive® VRM motherboard bulk capacitance specification
5	X	X	X	X	X	Doc	MTRR precedences and memory type overriding
6	X	X	X	X	X	Doc	V_{CC}S pins removed from specification
7					X	Doc	Split lock across cache line boundary disable bit added
8					X	Doc	Instruction streaming buffer disable bit added
9	X	X	X	X	X	Doc	System bus timings changes
10	X	X	X	X	X	Doc	Minimum storage specification change for processors with 1 MB L2 cache
11	X	X	X	X	X	Doc	WC buffer eviction data ordering

ERRATA

1. Mixed Cacheability of Lock Variables is Problematic in MP Systems

PROBLEM: This errata only affects multiprocessor systems where a lock variable address is marked cacheable in one processor and uncacheable in any others. The processors which have it marked uncacheable may stall indefinitely when accessing the lock variable. The stall is only encountered if:

- One processor has the lock variable cached, and is attempting to execute a cache lock.
- If the processor which has that address cached has it cached in the L2 only.
- Several other processors, meanwhile, issue back to back accesses to that same address on the bus.

IMPLICATION: MP systems where all processors either use cache locks or consistent locks to uncacheable space will not encounter this problem. If, however, a lock variable's cacheability varies in different processors, and several processors are all attempting to perform the lock simultaneously, an indefinite stall may be experienced by the processors which have it marked uncacheable in locking the variable (if the conditions above are satisfied). Intel has only encountered this problem in focus testing with artificially generated external events. Intel has not currently identified any commercial software which exhibits this problem.

WORKAROUND: Follow a homogenous model for the memory type range registers (MTRRs), ensuring that all processors have the same cacheability attributes for each region of memory; do not use locks whose memory type is cacheable on one processor, and uncacheable on others. Avoid page table aliasing, which may produce a nonhomogenous memory model.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

2. FP Data Operand Pointer May Be Incorrectly Calculated After FP Access Which Wraps 64-Kbyte Boundary in 16-Bit Code

PROBLEM: The FP Data Operand Pointer is the effective address of the operand associated with the last noncontrol floating-point instruction executed by the machine. If an 80-bit floating-point access (load or store) occurs in a 16-bit mode other than protected mode (in which case the access will produce a segment limit violation), the memory access wraps a 64-Kbyte boundary, and the floating-point environment is subsequently saved, the value contained in the FP Data Operand Pointer may be incorrect.

IMPLICATION: A 32-bit operating system running 16-bit floating-point code may encounter this erratum, under the following conditions:

- The operating system is using a segment greater than 64 Kbytes in size.
- An application is running in a 16-bit mode other than protected mode.
- An 80-bit floating-point load or store which wraps the 64-Kbyte boundary is executed.
- The operating system performs a floating point environment store (FSAVE/FNSAVE/FSTENV/FNSTENV) after the above memory access.
- The operating system uses the value contained in the FP Data Operand Pointer.

Wrapping an 80-bit floating-point load around a segment boundary in this way is not a normal programming practice. Intel has not currently identified any software which exhibits this behavior.

WORKAROUND: If the FP Data Operand Pointer is used in an OS which may run 16-bit floating-point code, care must be taken to ensure that no 80-bit floating-point accesses are wrapped around a 64-Kbyte boundary.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

3. Differences Exist in Debug Exception Reporting

PROBLEM: There exist some differences in the reporting of code and data breakpoint matches between that specified by previous Intel processors' specifications and the behavior of the Pentium Pro processor, as described below:

CASE 1:

The first case is for a breakpoint set on a MOVSS or POPSS instruction, when the instruction following it causes a debug register protection fault (DR7.gd is already set, enabling the fault). The Pentium processor reports delayed data breakpoint matches from the MOVSS or POPSS instructions by setting the matching DR6.bi bits, along with the debug register protection fault (DR6.bd). If additional breakpoint faults are matched during the call of the debug fault handler, the Pentium processor sets the breakpoint match bits (DR6.bi) to reflect the breakpoints matched by both the MOVSS or POPSS breakpoint and the debug fault handler call. The Pentium Pro processor only sets DR6.bd in either situation, and does not set any of the DR6.bi bits.

CASE 2:

In the second breakpoint reporting failure case, if a MOVSS or POPSS instruction with a data breakpoint is followed by a store to memory which crosses a 4-Kbyte page boundary, the breakpoint information for the MOVSS or POPSS will be lost. Previous processors retain this information across such a page split.

CASE 3:

If they occur after a MOVSS or POPSS instruction, the INT *n*, INTO, and INT3 instructions zero the DR6.Bi bits (bits B0 through B3), clearing pending breakpoint information, unlike previous processors.

CASE 4:

If a data breakpoint and an SMI (System Management Interrupt) occur simultaneously, the SMI will be serviced via a call to the SMM handler, and the pending breakpoint will be lost.

CASE 5:

When an instruction which accesses a debug register is executed, and a breakpoint is encountered on the instruction, the breakpoint is reported twice.

IMPLICATION: When debugging or when developing debuggers for a Pentium Pro processor system, this behavior should be noted. Normal usage of the MOVSS or POPSS instructions (i.e., following them with a MOV ESP) will not exhibit the behavior of cases 1-3. Debugging in conjunction with SMM will be limited by case 4 (no workaround has been identified for this case).

WORKAROUND: Following MOVSS and POPSS instructions with a MOV ESP instruction when using breakpoints will avoid the first three cases of this erratum. There are no known workarounds for cases 4 or 5.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

4. FLUSH# Servicing Delayed While Waiting for STARTUP_IPI in MP Systems

PROBLEM: In a multiprocessor system, if an application processor is waiting for a startup interprocessor interrupt (STARTUP_IPI), then it will not service a FLUSH# pin assertion until it has received the STARTUP_IPI.

IMPLICATION: After the MP initialization protocol, only one processor becomes the bootstrap processor (BSP). All others become slave application processors (APs). After losing the BSP arbitration, APs go into a wait loop waiting for a STARTUP_IPI.

The BSP can wake up an AP to perform some tasks with a STARTUP_IPI, and then put one or more APs back to sleep with an initialization interprocessor interrupt (INIT_IPI, which has the same affect as asserting INIT#), which returns them to a wait loop. The result is a possible loss of cache coherency if the offline processor is intended to service a FLUSH# assertion at this point. The FLUSH# will be serviced as soon as the processor is awakened by a STARTUP_IPI, before any other instructions are executed. Intel has not encountered any operating systems that are affected by this problem.

WORKAROUND: Operating system developers should take care to execute a WBINVD instruction before an AP is taken offline using an INIT_IPI.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

5. Fast Strings REP MOVS May Not Transfer All Data

PROBLEM: If the fast strings feature is enabled (bit 2 of MSR 1E0h is 0), a REP MOVS instruction may incorrectly use a pre-decremented counter, thereby skipping a cache line of the transfer. This will only occur if paging is enabled.

IMPLICATION: A REP MOVS instruction executed with this feature enabled may not transfer all requested data, leading to stale data being left at the destination.

WORKAROUND: For steppings affected by this erratum, the fast strings feature must be disabled in the BIOS, by setting bit 2 of the machine specific register (MSR) at address 1E0h to '1'.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

6. Page Table Base Change During Task Switch Using Mode C Paging May Corrupt EIP

PROBLEM: While using mode C paging (a 36-bit address extension which uses 2-Mbyte pages), if data is written to CR3 during a task switch, the EIP will be corrupted, resulting in unpredictable failure. CR3, which contains the page table base, will be written during a task switch if the switch is moving to a new paging environment, or if the TLB entries must be cleared.

IMPLICATION: An operating system which uses a hardware tasking model (via the task switch segment, or TSS) along with the 36-bit extension mode C paging, may fail, most likely with an invalid opcode exception. Intel has not identified any operating systems which use this combination of features.

WORKAROUND: When developing an operating system which uses mode C paging, do not use hardware task switching, or else only perform hardware task switches which do not change the page table base register, CR3.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

7. Code Fetch Matching Disabled Debug Register May Cause Debug Exception

PROBLEM: The bits L0-3 and G0-3 enable breakpoints local to a task and global to all tasks, respectively. If one of these bits are set, a breakpoint is enabled, corresponding to the addresses in the debug registers DR0-DR3. If at least one of these breakpoints are enabled, any of these registers are *disabled* (i.e., L*n* and G*n* are 0), and RW*n* for the disabled register is 00 (indicating a breakpoint on instruction execution), normally no instruction fetch will cause an instruction-breakpoint fault based on a match with the address in the disabled register(s). However, if the address in a disabled register matches the address of a code fetch which also results in a page fault, an instruction-breakpoint fault will occur.

IMPLICATION: While debugging software, extraneous instruction-breakpoint faults may be encountered if breakpoint registers are not cleared when they are disabled. Debug software which does not implement a code breakpoint handler will fail, if this occurs. If a handler is present, the fault will be serviced. Mixing data and code may exacerbate this problem by allowing disabled data breakpoint registers to break on an instruction fetch.

WORKAROUND: The debug handler should clear breakpoint registers which become disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

8. Mode C Paging in SMM Causes Use of Incorrect Page Tables

PROBLEM: If mode C paging (a 36-bit address extension which uses 2-Mbyte pages) is in use, an SMI is serviced, and the SMM handler switches to mode C paging with a different set of page tables, then upon returning from the SMM handler via an RSM instruction, the processor will use the wrong page tables for address translation. Since paging is not, in general, supported in SMM, this is not a violation of the Pentium Pro processor specification, and has been removed as an erratum, but has been included here for continuity.

IMPLICATION: Using mode C paging both in normal operation and in System Management Mode (SMM) may cause the system to hang. Intel has not currently identified any software which is affected by this erratum. This is not a violation of the Pentium Pro processor specification; large size pages are not supported in SMM.

WORKAROUND: Do not use mode C paging for the SMM handler code.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

9. Memory Indirect Near Call May Corrupt EIP

PROBLEM: Under some conditions, a near indirect call (opcode FF /2) may cause corruption of the EIP after partial completion of the instruction's execution. This can occur if an access due to the near indirect call:

- Accesses memory which crosses a page boundary.
- Results in a misaligned data breakpoint exception.
- Is the first access of the page containing the data used for the call.
- Results in a page fault which, upon walking the page tables (see Erratum 13), is found not to be a faulting condition.

IMPLICATION: The most likely failure mechanism for this erratum is an invalid opcode exception.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

10. L2 Single-Bit Correctable Error May Cancel Simultaneous Valid Data

PROBLEM: If a single-bit correctable error is detected by the Pentium Pro processor on a transaction between the data cache unit (DCU) of the CPU and the L2 cache, and there is a second transaction pending which caches data from the Pentium Pro processor bus, the Pentium Pro processor will cancel both transactions instead of just the transaction on which the error occurred.

IMPLICATION: The canceled transaction will never complete under these conditions. Single-bit errors which encounter this boundary condition will cause the Pentium Pro processor to hang. However, since this erratum only affects error recovery, normal, hardware error-free execution will not be affected.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

11. Page Split Access Before Write to CR3 May Cause Hang

PROBLEM: If paging is enabled, and a memory access occurs which crosses a page boundary, a future write to CR3 may cause the processor to hang. The write to CR3 must occur at a specific time interval away from the page split access; in focus-testing, the only test which encountered this hang in mode A or B paging had 12 NOP instructions between the split access and the write to CR3. When using mode C paging (2-Mbyte pages), the probability of encountering this erratum increased somewhat.

IMPLICATION: Using paging in conjunction with writes to CR3 (to invalidate the page tables or change to a new paging environment) may cause the system to hang. There is a small, timing dependent window of opportunity to encounter this erratum, which is wider if mode C paging is used.

WORKAROUND: Some instructions can be inserted between the memory access and the write to CR3 to prevent this erratum, such as:

- CPUID,
- STD,
- CLD, or
- CLI/STI.

Though the following instructions between the memory access and the write to CR3 prevent the erratum, they may not be suitable for all situations as workarounds for this erratum due to their effect or dependency on system state:

- WRMSR,
- RDMSR,
- RDPMC,
- LDS, LFS, LGS, LSS, or LES,
- MOV CR0, reg,
- MOV CR2, reg,
- MOV CR4, reg,
- MOV DS, reg,
- MOV SS, reg,
- MOV ES, reg,

- MOV GS, reg,
- MOV FS, reg,
- A far transfer, interrupt, or software interrupt.

Any of these which occur between the memory access which crosses a 4-Kbyte boundary and the write to CR3 will prevent the conditions necessary for this erratum to occur. Due to its long latency, a nonstring OUT instruction will also prevent the erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

12. Active A20M# During SMM Dump

PROBLEM: If the A20M# signal (mask A20) is active when the SMM handler is saving CPU state information to the SMM dump space prior to executing the handler code, the state information will be partially saved at an address with the A20 pin masked, and partially with A20 unmasked.

IMPLICATION: If the SMM dump space uses addresses in an odd megabyte (A20 high), the stored state will be split among two different address spaces and the system will not recover state correctly after an RSM instruction.

WORKAROUND: Either ensure that the SMM dump space does not coincide with any odd-megabyte addresses (under 1 Mbyte is acceptable), or use external logic to prevent the servicing of an SMI until A20M# is deasserted.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

13. Split Access Across 4-Kbyte Page Boundary May Cause Hang

PROBLEM: When a load or store access is split across a page boundary, a page fault may be detected. If a page fault is signaled during such an access, the Pentium Pro processor will walk the page table to ensure that the page is not being made accessible by another agent (such as another processor in an MP system, or a DMA controller) before branching to the page fault handler. If the split access signals a page fault, and the page which caused the fault is then found to be nonfaulting during the page table walk, the access will be executed incorrectly, resulting in a hang.

IMPLICATION: When multiple agents are capable of changing the page table, a load or store which is split across a page boundary may cause the machine to hang.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

14. Checker BIST Failure in FRC Mode Not Signaled

PROBLEM: If a system is running in functional redundancy checking (FRC) mode, and the checker of the master-checker pair encounters a hard failure while running the built-in self test (BIST), the checker will tri-state all outputs without signaling an IERR#.

IMPLICATION: Assuming the master passes BIST successfully, it will continue execution unchecked, operating without functional redundancy. However, the necessary pull-up on the FRCERR pin will cause an FRCERR to be signaled. The operation of the master depends on the implementation of FRCERR.

WORKAROUND: For successful detection of BIST failure in the checker of an FRC pair, the FRCERR signal must be used, instead of IERR#.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

15. BINIT# Assertion Causes FRCERR Assertion in FRC Mode

PROBLEM: If a pair of Pentium Pro processors are running in functional redundancy checking (FRC) mode, and a catastrophic error condition causes BINIT# to be asserted, the checker in the master-checker pair will enter shutdown. The next bus transaction from the master will then result in the assertion of FRCERR.

IMPLICATION: Bus initialization via an assertion of BINIT# occurs as the result of a catastrophic error condition which precludes the continuing reliable execution of the system. Under normal circumstances, the master-checker pair would remain synchronized in the execution of the BINIT# handler. However, due to this erratum, an FRCERR will be signaled. System behavior then depends on the implementation of FRCERR.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

16. Extra Page Fault May Occur on IRET During Task Switch

PROBLEM: During a hardware task switch (using the task switch segment, or TSS) which jumps to the new task with an IRET which causes a page fault, a second page fault exception may be generated upon completion of the page miss handler routine, prompting the Pentium Pro processor to execute the page miss handler a second time.

IMPLICATION: Intel has identified no adverse implications of this anomaly other than the slight performance loss from executing the page miss handler twice after a task switch which causes a page fault.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

17. Some Caching Models in SMM May Cause Shutdown

PROBLEM: A condition exists in the Pentium Pro processor such that upon returning from the SMM handler (via an RSM instruction), a read may be executed too soon for the data to be successfully loaded. This condition occurs whenever the load for the read is delayed because the cache line which the load accesses is not present in the cache. This may occur if:

- The SMM handler is too large to fit into the cache.
- A WBINVD instruction was executed upon servicing the SMI.
- The SMM handler is being run in memory mapped with the UC (uncacheable) memory type.

When the RSM instruction is executed under any of these conditions, the first memory access after the RSM will cause multiple stack faults, and the Pentium Pro processor will enter shutdown.

IMPLICATION: Under many operating conditions, returning from SMM will result in the CPU entering shutdown.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

18. Fast Strings Feature Re-enabled After INIT Event

PROBLEM: If the Fast Strings feature has been disabled in the machine specific registers (MSRs) by clearing the appropriate bit (to '0'), after an INIT or an initialization interprocessor interrupt (INIT_IPI), the bit will be set (to '1'), and the feature will be re-enabled.

IMPLICATION: After each INIT or INIT_IPI event, this feature will be re-enabled, exposing the system to the effects of Erratum 5.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

19. THERMTRIP# Feature Not Present

PROBLEM: The THERMTRIP# feature documented for the Pentium Pro processor is not present.

IMPLICATION: The Pentium Pro processor will not shut down upon reaching a catastrophic thermal condition as defined in the *Pentium® Pro Processor Developer's Manual, Volume 1.*

WORKAROUND: System manufacturers can implement a thermal management solution which does not rely upon THERMTRIP#.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

20. OUT Instruction, Branch Trace Message May Write Incorrect Data

PROBLEM: A data cache load may be blocked by internal conditions, and speculatively awakened by the Pentium Pro processor. If this speculative wakeup occurs at the same time that an OUT instruction or a branch trace message (BTM) is dispatched, and the load access crosses a cache line boundary, the buffer which contains the data for the OUT or BTM will be corrupted.

IMPLICATION: Some OUT instructions and BTMs may propagate corrupted data.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

21. THERMTRIP# Pin Not Asserted For Catastrophic Thermal Condition

PROBLEM: If a catastrophic thermal condition is reached as specified in the *Pentium® Pro Processor Developer's Manual, Volume 1,* the processor will correctly shut down until a hard reset is performed. However, the THERMTRIP# pin, which should be asserted after the processor shuts down, will remain unasserted.

IMPLICATION: If the processor shuts down due to a catastrophic thermal condition, there will be no electrical indication of the reason for the shutdown, since the THERMTRIP# pin will not be asserted.

WORKAROUND: An external temperature sensor should be implemented if a system design requires notification that a catastrophic thermal condition has caused the system to shut down.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

22. LBER Value May Not Be Updated Correctly

PROBLEM: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after executing a trap under some conditions, the LBER may not be updated after a branch, or may be updated incorrectly.

IMPLICATION: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, the value of LBER is 0, or the value of the LBER is the same before and after the trap, then the LBER value is incorrect.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

23. BTM for SMI Will Contain Incorrect FROM EIP

PROBLEM: A system management interrupt (SMI) will produce a branch trace message (BTM), if BTMs are enabled. However, the FROM EIP field of the BTM (used to determine the address of the instruction which was being executed when the SMI was serviced) will not have been updated for the SMI, so the field will report the same FROM EIP as the previous BTM.

IMPLICATION: A BTM which is issued for an SMI will not contain the correct FROM EIP, limiting the usefulness of BTMs for debugging software in conjunction with System Management Mode (SMM).

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

24. Task Switch Fault May Allow Read Access of Linear Address 0h

PROBLEM: For this erratum to occur, the following conditions must be satisfied:

- A Task State Segment (TSS) must be used to handle task switches.
- A task switch fault must occur while loading the new segment descriptors for a task switch.
- The resulting task switch fault handler call must not result in a task switch.
- The task switch fault handler must be executed successfully and return to the task switch using a far return or an IRET instruction.

Upon the coincidence of these conditions, any of the data segment registers which were not loaded by the task switch will permit a read access to a single byte at linear address 0h.

IMPLICATION: Under these circumstances, if the target task is of a privilege level of 1, 2, or 3, it may be allowed to read the byte at linear address 0h, which it may not normally be permitted to read. Intel has not currently identified any software which could cause this behavior.

WORKAROUND: When developing an operating system which uses the TSS to process task switches, a task switch fault should also generate a task switch using the TSS.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

25. Low Frequencies With 5:2 Core to Bus Clock Ratio May Fail in FRC

PROBLEM: If a functional redundancy checking (FRC) system is using a 5:2 core to bus clock ratio, and is running at frequencies of 120/48 MHz or lower, the phase-locked loops (PLL) of the master-checker pair may not function correctly, resulting in the assertion of FRCERR with the first request to the Pentium Pro processor bus. If the first request does not result in the assertion of FRCERR, the system will function normally. The minimum core frequency specification in the *Pentium® Pro Processor Developer's Manual*, *Volume 1,* is 100 MHz.

IMPLICATION: An FRC system should not be run at core frequencies lower than 120 MHz with a 5:2 core to bus clock ratio. If it is necessary to do so, the system may not always begin executing correctly, but power-cycling the system after such behavior may allow normal operation.

WORKAROUND: Run the system at its specified frequency, or power-cycle the system if it is necessary to run an FRC system under these conditions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

26. RDPMC Cannot Be Used in Conjunction With SMM

PROBLEM: If the performance counter enable (PCE) bit (bit 8 of CR4) is enabled (set to '1'), and if an RSM instruction is executed, the processor will enter a shutdown state, believing that reserved space has been altered.

IMPLICATION: The PCE bit is set by an operating system to enable user code to issue an RDPMC (Read Performance Monitor Counter) instruction. In systems which support SMM, this bit should not be set, and this instruction will not be available. However, the performance monitoring capabilities of the Pentium Pro processor are still available through driver-based applications.

WORKAROUND: To prevent the shutdown from occurring, SMM handler code can clear this bit. However, this will disable the ability of users to issue the RDPMC instruction, and any application which attempts to issue this instruction after SMM handler code has been executed will cause a general protection fault, unless the operating system has re-enabled it by setting the PCE bit.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

27. PWRGOOD Forced to 0 During Boundary Scan Resets TAP

PROBLEM: According to IEEE 1149.1-1990 (JTAG) Specification, rule 5.3.1(b), no component input pin should initialize the test access port (TAP) controller, or any other test logic within the component. However, if the PWRGOOD pin on the Pentium Pro processor is forced to 0, it will reset the TAP Controller.

IMPLICATION: If boundary scan testing forces PWRGOOD to 0, boundary scan errors will result.

WORKAROUND: Do not force PWRGOOD to 0 during boundary scan testing. The BSDL file or the test vectors can be edited to ensure that this does not occur, by changing the declaration of the PWRGOOD pin from 'input' to 'internal' with a safe value of 1. Note that while this will allow boundary scan tests to be generated correctly, it will not allow the PWRGOOD pin to be tested.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

28. BIST Failure Not Indicated When RUNBIST TAP Command Used

PROBLEM: The test access port (TAP) RUNBIST command is intended for use during boundary scan testing, and should run the built-in self test (BIST), storing the result in the BIST result register after completion. A zero

result indicates BIST passed, and a nonzero result indicates BIST failure. Currently, if this command is executed from the TAP, a zero result will be stored in the BIST result register regardless of whether the BIST passes or fails.

IMPLICATION: Running the RUNBIST command from the TAP will not indicate a failing condition. This command is not functional for the steppings of the Pentium Pro processor affected by this erratum.

WORKAROUND: None identified. However, BIST may still be run separately from boundary scan, by allowing the INIT# signal to be sampled active on the RESET# signal's active to inactive transition. In this case, passage or failure is indicated in EAX.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

29. INVLPG May Not Invalidate Targeted ITLB Entry

PROBLEM: When the INVLPG instruction is used, the entry in the translation look-aside buffer (TLB) which corresponds to the given address should be invalidated, under most circumstances preserving the validity of the rest of the TLB. However, if the variable memory type range registers (MTRRs) are used to map a region of memory as a particular memory type, this may not occur. Specifically, if the variable MTRR specifies a range which is not 4-Mbyte aligned (i.e., the MTRR's mask register contains a value with one or more bits in the range [21:12] nonzero), the INVLPG instruction will invalidate the proper entry in the data TLB (DTLB), but in so doing, may lose the information necessary to invalidate the entry in the instruction TLB (ITLB). It will, instead, invalidate entry 0 in the ITLB, leaving a stale translation in the ITLB. A subsequent code fetch from the addresses in this entry will retrieve and execute incorrect instructions.

IMPLICATION: The INVLPG entry cannot be used on a system with an MTRR range which is not 4-Mbyte aligned.

WORKAROUND: BIOS and operating system code should not use a variable MTRR range which is not 4-Mbyte aligned. All such ranges should have bits [21:12] of the MTRR's mask register set to '0' for processor steppings affected by this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

30. SMI Does Not Flush TLB Entries With PGE Enabled

PROBLEM: When SMI# is asserted, all entries of the translation lookaside buffer (TLB) should be flushed. However, entries with the page global enable (PGE) bit set will not be flushed from the TLB.

IMPLICATION: The SMM (System Management Mode) handler may access an address which will be incorrectly translated due to the stale entries in the TLB, resulting in unexpected system behavior.

WORKAROUND: If a write to CR3 is executed to clear the TLBs upon entry into SMM, then BIOS code can contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

31. I/O Restart in SMM May Fail After Simultaneous MCE

PROBLEM: If an I/O instruction (IN, INS, REP INS, OUT, OUTS, or REP OUTS) is being executed, and if the data for this instruction becomes corrupted, the Pentium Pro processor will signal a machine check exception (MCE). If the instruction is directed at a device which is powered down, the processor may also receive an assertion of SMI#. Since MCEs have higher priority, the processor will call the MCE handler, and the SMI# assertion will remain pending. However, upon attempting to execute the first instruction of the MCE handler, the

SMI# will be recognized and the processor will attempt to execute the SMM handler. If the SMM handler is completed successfully, it will attempt to restart the I/O instruction, but will not have the correct machine state, due to the call to the MCE handler.

IMPLICATION: A simultaneous MCE and SMI# assertion may occur for one of the I/O instructions above. The SMM handler may attempt to restart such an I/O instruction, but will have corrupted state due to the MCE handler call, leading to failure of the restart and SHUTDOWN of the processor.

WORKAROUND: If a system implementation must support both SMM and MCEs, the first thing the SMM handler code (when an I/O restart is to be performed) should do is check for a pending MCE. If there is an MCE pending, the SMM handler should immediately exit via an RSM instruction and allow the machine check exception handler to execute. If there is not, the SMM handler may proceed with its normal operation.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

32. SMBASE Reset on INIT# or INIT_IPI

PROBLEM: In systems which support System Management Mode (SMM), the default value for the SMBASE is 30000h. This value may be changed, particularly in MP systems, where processors' SMM spaces do not overlap. However, when an assertion of the INIT# pin or an initialization interprocessor interrupt (INIT_IPI) is received by a Pentium Pro processor, the value of SMBASE is reset to 30000h.

IMPLICATION: An assertion of INIT# or an INIT_IPI will clear any changes made to the SMBASE value.

WORKAROUND: It is possible that BIOS code may contain a workaround for this erratum, in systems which contain C0 or subsequent steppings of Pentium Pro processor silicon. No workaround is available for the B0 stepping of the Pentium Pro processor.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

33. MCE Due to L2 Cache Error Gives L1 MCACOD.LL

PROBLEM: If an error occurs on an access to the Pentium Pro processor's L2 cache, any resulting Machine Check Exception (MCE) will be logged with '01' in the LL field of MCACOD. This value indicates an L1 cache error; the value should be '10'.

IMPLICATION: An MCE due to an L2 cache access will be improperly logged as due to an L1 error.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

34. INVLPG Does Not Invalidate Entire 0 to 4-Mbyte Region

PROBLEM: When using large paging modes (mode C or mode B paging), a TLB entry may be created for the large page beginning at address 0. This page may cover the 0 to 2 or 4-Mbyte region. If the fixed memory type range registers (MTRRs) are used in conjunction with these paging modes, TLB entries for 4-Kbyte pages may also be created to map the lower 1-Mbyte region of memory. If the INVLPG instruction is used to invalidate one of these 4-Kbyte pages, the large page which covers the same region will not be invalidated; conversely, if the large page is invalidated using INVLPG (with an address from 1 to 2 or 4 Mbytes), the 4-Kbyte entries will not be invalidated.

IMPLICATION: Intel has not currently identified any software which is affected by this erratum. However, using INVLPG to invalidate regions from 0 to 1 Mbyte or from 1 to 4 Mbytes may result in the use of stale data if the other part of the 0 to 4-Mbyte range is expected to be invalidated when using 4-Mbyte pages (or 0 to 2-Mbyte if

the software uses 2-Mbyte pages). Future operating systems may be affected by this erratum, but only if the 0 to 4-Mbyte page is invalidated using INVLPG. Since this page would most likely contain operating system kernel code, it is unlikely that such an operating system would be designed to invalidate this page.

WORKAROUND: Operating systems should not set up a 4-Mbyte page at address 0, unless writes to the CR3 register are performed to clear the TLB entries for this region (if the PGE bit in CR4 is set, this bit must be cleared prior to writing to CR3, to clear global pages). This will invalidate the entire TLB. If a 4-Mbyte page is used, and INVLPG instructions are used to invalidate this region, INVLPG instructions must be used repeatedly to invalidate the large page as well as any of the 4-Kbyte ranges which should be invalidated.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

35. BINIT# Assertion During Snoop Hit May Cause Double Machine Check Exception

PROBLEM: If the L1 cache has locked a modified cache line in response to an external snoop, then the assertion of BINIT# between the time ADS# is asserted for the snoop and the time that the implicit writeback data is transferred may cause a double machine check exception (MCE). The double MCE is encountered due to artificial internal parity errors generated by the BINIT# handler during its cleanup routines.

IMPLICATION: A BINIT# may cause the processor to shut down in response to the double MCE generated if the above conditions occur. IERR# will be asserted, and no further operations are possible until a hard reset is performed.

WORKAROUND: BINIT# is asserted upon detection of a catastrophic system condition. Leaving MCEs disabled will result in the BINIT# causing the CPU to enter shutdown as well. To avoid CPU shutdown, BINIT# observation can be disabled. However, a condition which would result in the generation of BINIT# may cause corrupt data to be propagated, and unexpected system behavior may occur. No workaround which would enable BINIT# to be recognized in this window without causing the CPU to shutdown has been identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

36. Machine Check Exception Handler May Not Always Execute Successfully

PROBLEM: An asynchronous machine check exception (MCE), such as a BINIT# event, which occurs during an access that splits a 4-Kbyte page boundary may leave some internal registers in an indeterminate state. Thus, MCE handler code may not always run successfully if an asynchronous MCE has occurred previously.

IMPLICATION: An MCE may not always result in the successful execution of the MCE handler, hanging the CPU.

WORKAROUND: If MCEs are not enabled, execution of the MCE handler will not occur. However, asynchronous MCEs usually occur upon detection of a catastrophic system condition. Leaving MCEs disabled will result in the condition which caused the asynchronous MCE instead causing the CPU to enter shutdown. Therefore, leaving MCEs disabled may not improve overall system behavior. No workaround which would guarantee successful MCE handler execution under this condition has been identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

37. Double ECC Error on Read May Result in BINIT#

PROBLEM: For this erratum to occur, the following conditions must be met:

- Machine check exceptions (MCEs) must be enabled.
- A dataless transaction (such as a write invalidate) must be occurring simultaneously with a transaction which returns data (a normal read).
- The read data must contain a double-bit uncorrectable ECC error.

If these conditions are met, the Pentium Pro processor will not be able to determine which transaction was erroneous, and instead of generating an MCE, it will generate a BINIT#.

IMPLICATION: The bus will be reinitialized in this case. However, since a double-bit uncorrectable ECC error occurred on the read, the MCE handler (which is normally reached on a double-bit uncorrectable ECC error for a read) would most likely cause the same BINIT# event.

WORKAROUND: Though the ability to drive BINIT# can be disabled in the Pentium Pro processor, which would prevent the effects of this erratum, overall system behavior would not improve, since the error which would normally cause a BINIT# would instead cause the machine to shut down. No other workaround has been identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

38. RSM Cannot Return to HALT or SHUTDOWN in 32-Bit OS

PROBLEM: If a processor which was previously operating in 32-bit protected mode has shut down or executed a HLT instruction, and is then targeted by a System Management Interrupt (SMI), the SMM handler will be executed in a different mode. Upon returning to the HALT or SHUTDOWN state via an RSM, the mode state will not be correctly updated back to 32-bit protected mode, potentially crashing the operating system.

IMPLICATION: Systems which support SMM may hang if software which executes HLT instructions is being run, or if the processor is expected to be able to recover from a SHUTDOWN state via software recovery which uses SMM.

WORKAROUND: For the HALT case, SMM handler code may clear the Halt Auto Restart bit in the SMM dump space (offset 7F02h), then subtract 1 from the saved EIP (offset 7FF0h). This will cause the HLT instruction to be re-executed upon completion of the RSM, which will allow 32-bit protected mode to be re-entered and the processor to successfully enter the HALT state. No workaround has currently been identified for the SHUTDOWN case; however, it is not guaranteed that an SMI will be properly serviced when in this state.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

39. Accesses of Modified Data in DCU May Hang System

PROBLEM: Under heavy bus traffic in a system with multiple snooping agents (including processors or DMA controllers) which are accessing multiple addresses that map to the same set of the data cache unit (DCU) and L2 cache, the following sequence of transactions may cause the system to hang:

1. A read-for-ownership cycle (a.k.a read and invalidate line) is issued to an address which is in Modified state in both the L2 cache and the DCU.
2. The DCU writes the data at that address back to the L2 cache, so that it is no longer in Modified state in the DCU, only in the L2 cache. It is now in Invalidated state in the DCU.
3. The DCU issues a read to the same address in the L2 cache, and the data is returned in Exclusive state (which is incorrect).
4. The L2 cache proceeds to evict its copy of the line in Modified state. Meanwhile, the DCU modifies its copy of the line in Exclusive state, which then becomes Modified.

5. An external snoop from another processor or DMA controller occurs to the address, receives a HITM# (hit Modified) response, and the DCU's data goes out onto the processor bus.

At this stage, the internal state of the processor may be incorrect, due to an incomplete L2 cache eviction. If the DCU attempts to access the line again, or if another external snoop occurs to the line, the processor will hang, never completing the operation.

IMPLICATION: Heavy traffic in a system with devices capable of causing external snoops may cause the system to hang. However, Intel has observed the conditions necessary to cause the traffic described above only on 3- or 4-way MP systems, under heavy traffic or stress-testing conditions where same-set data is being rapidly transferred between main memory, the L2 cache, and the DCU.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

40. Branch Traps Do Not Function if BTMs Are Also Enabled

PROBLEM: If branch traps or branch trace messages (BTMs) are enabled alone, both function as expected. However, if both are enabled, only the BTMs will function, and the branch traps will be ignored.

IMPLICATION: These debugging features cannot be used together.

WORKAROUND: If branch trap functionality is desired, BTMs must be disabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

41. Cache Line May Exist in Two Different Ways in MP System

PROBLEM: In heavy traffic on an MP system, it is possible for a cache line to be read into two different "ways" of the L2 cache, or for an invalid line to become spuriously validated. The conditions necessary for this behavior to occur are as follows:

- A read-for-ownership (a.k.a. read and invalidate line), normal read, and second read-for-ownership must occur, in that order, with no other intervening transactions.
- These three transactions must be pipelined together.
- They must be accessing the same set of the L2 cache.
- A cache line which is in the process of being changed to Shared state must be snooped by an external agent (such as another processor or DMA controller).

IMPLICATION: If these conditions occur, unexpected behavior will result. The most likely failure mechanism for this erratum is a processor hang, but it is possible that data corruption may result. Intel has observed this erratum only in a focus-testing environment with 3 or more processors in the system. Intel has not currently identified any software which causes this behavior.

WORKAROUND: Intel has been unable to reproduce this erratum with an IOQ depth of 1, as configured by the BIOS. This is a recommended configuration for the affected steppings of the processor; production systems with this configuration are not known to be affected by this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

42. HALT, SHUTDOWN, and STPCLK Special Cycles Not Issued

PROBLEM: The HALT, SHUTDOWN, and STPCLK special cycles will not be issued by the Pentium Pro processor.

IMPLICATION: Systems which rely on these special cycles will not function correctly. Software that purposely creates a processor SHUTDOWN condition in order to create a processor restart will SHUTDOWN without the restart occurring.

WORKAROUND: No known workaround exists for systems which use the B0 stepping of the Pentium Pro processor or the A2 stepping of the Intel 450KX PCIset. Other systems may contain a workaround for this erratum in BIOS code.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

43. L2 Cache May Incorrectly Report BIST Failure

PROBLEM: If BIST (Built-In Self Test) is requested on reset by keeping INIT# asserted past the rising edge of RESET# (i.e., by performing a normal BIST request), it is possible that the L2 cache BIST will not be run. If this occurs, the BIST result reported in the EAX register upon completion of BIST will contain the signature for an L2 cache failure (bits 2, 16, and/or 17 may be nonzero).

IMPLICATION: A BIST failure result from the L2 cache may not be valid. This invalid failure result becomes more probable as the RESET# pulse is lengthened. This erratum has been observed by Intel on devices with both 256-Kbyte and 512-Kbyte L2 caches. This was fixed in the β steppings of the 256-Kbyte and 512-Kbyte L2 caches.

WORKAROUND: Ignore the result of the L2 BIST by masking bits 2, 17, and 16 of the EAX register before checking the BIST result, or do not run BIST.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

44. LOCK Prefix Does Not Generate Trap With Some Instructions

PROBLEM: If the LOCK prefix is used with a CPUID or a near call indirect (FF/2 type of CALL) instruction, a #UD exception should be generated in protected or virtual 8086 mode, and an interrupt 6 should be generated in real address mode. On the affected steppings of the Pentium Pro processor, these exceptions may not occur; the processor may instead decode and execute the instruction as if the LOCK prefix was not present.

IMPLICATION: Illegal use of the LOCK prefix may not always be detected. In cases where it is not detected, it will be ignored (the LOCK# signal will not be asserted for that instruction).

WORKAROUND: Do not use the LOCK prefix with the CALL or CPUID instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

45. FRCERR May Be Incorrectly Asserted on Explicit Writeback Cycles

PROBLEM: In a system configured for FRC (Functional Redundancy Checking) mode, an explicit writeback operation (due to a capacity miss in the L2 cache, for example) may cause FRCERR to be asserted, even though no mismatch between the master and checker processors have occurred.

IMPLICATION: For the affected steppings of the Pentium Pro processor, FRC mode is not reliable when used in conjunction with explicit writebacks.

WORKAROUND: If the system is configured to run with all memory in write-through mode, explicit writebacks will not occur (because all writes go through to the processor bus), thus avoiding this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

46. FP Inexact-Result Exception Flag May Not Be Set

PROBLEM: When the result of a floating-point operation is not exactly representable in the destination format (1/3 in binary form, for example), an inexact-result (precision) exception occurs. When this occurs, the PE bit (bit 5 of the FPU status word) is normally set by the processor. Under certain rare conditions, this bit may not be set when this rounding occurs. However, other actions taken by the processor (invoking the software exception handler if the exception is unmasked) are not affected. This erratum can only occur if the floating-point operation which causes the precision exception is immediately followed by one of the following instructions:

- FST m32real
- FST m64real
- FSTP m32real
- FSTP m64real
- FSTP m80real
- FIST m16int
- FIST m32int
- FISTP m16int
- FISTP m32int
- FISTP m64int

Note that even if this combination of instructions is encountered, there is also a dependency on the internal pipelining and execution state of both instructions in the processor.

IMPLICATION: Inexact-result exceptions are commonly masked or ignored by applications, as it happens frequently, and produces a rounded result acceptable to most applications. The PE bit of the FPU status word may not always be set upon receiving an inexact-result exception. Thus, if these exceptions are unmasked, a floating-point error exception handler may not recognize that a precision exception occurred. Note that this is a "sticky" bit, i.e., once set by an inexact-result condition, it remains set until cleared by software.

WORKAROUND: This condition can be avoided by inserting two NOP instructions between the two floating-point instructions.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

47. Snoop Hit During RSM Segment Load May Hang System

PROBLEM: During the execution of an RSM instruction, the processor must reload the state which was saved into SMM (System Management Mode) dump space upon recognition of an SMI# event. If, while loading from an address in SMM memory which contains the saved contents of the segment registers (CS, DS, ES, FS, and GS), a snoop occurs from an external agent which matches the address (in an overlayed system memory region), then the resulting postponement and retry of the segment register load may result in the processor hanging due to an infinite loop in the microcode.

IMPLICATION: The processor will hang when exiting from SMM, if an address match occurs between an external snoop to memory and the address of a segment register being loaded from SMM dump space. Note that there is a very small window of opportunity for this erratum to occur.

WORKAROUND: Do not overlay system memory with the SMM dump space. If overlaying memory in this way is desired, ensure that the system does not snoop system memory overlaid with SMRAM (by making this region uncacheable, for example).

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

48. Mbyte-Aligned SMM Base Address Causes SHUTDOWN

PROBLEM: When servicing an SMI# event, the CS (code segment) selector is formed by shifting the bit value of the SMM (System Management Mode) base address field right by 4 bits and using the low order 16 bits for the selector. If the SMM base address is aligned on a megabyte boundary (i.e., with an SMM base address of 1, 2, 3, etc., Mbytes), these bits will all be 0, resulting in a segment selector value of 0000h, also known as a null selector. This results in the null bit for this segment being set in the segment register. If the CS selector containing the null value is subsequently used as a data selector while in SMM, a GP fault will result; since no interrupt table exists by default in SMM, the processor will then enter SHUTDOWN. Note that if the SMM base address is below 1 Mbyte, this erratum will not occur.

IMPLICATION: SMM base addresses which fall on a megabyte boundary (i.e., SMM_BASE mod 1048576 = 0) will result in system failure upon use of CS as a data selector in SMM. SMM base addresses below 1 Mbyte (most systems which support SMM are configured this way) will not be affected by this erratum.

WORKAROUND: If the SMM memory space must be placed above 1 Mbyte, do not use a Mbyte-aligned SMM base address when setting up SMM in the BIOS.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

49. Machine Check Exception with Coexisting Large and Small Pages

PROBLEM: The Pentium Pro processor has separate translation lookaside buffers (TLBs) for caching 4-Kbyte and 4-Mbyte page translations. When a single linear address is used to access data, and the processor contains an entry in both the 4-Kbyte and the 4-Mbyte TLBs for that address, an unrecoverable Machine Check Exception (MCE) is generated if MCEs are enabled in the system.

IMPLICATION: An unrecoverable MCE may be generated in an operating system environment which configures large pages after MCEs have been enabled (including cases where MCEs are enabled, an INIT# sequence is executed, and then large pages are configured).

WORKAROUND: This erratum may be avoided by ensuring that after the large page directory entries (PDEs) are created (e.g., by an operating system), all of the TLB entries are flushed before executing code which enables MCEs. Also, code which dynamically creates large pages should ensure that the large page PDEs are not mapped by small (4-Kbyte) PDEs. A BIOS which may cause an INIT# sequence must clear the MC1_CTL-MC4_CTL registers as part of the initialization sequence.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

50. WBINVD to Some Modified Addresses Hangs System

PROBLEM: If a system with cacheable memory in the address range 70000000h-7003FFFFh has a Modified address in that range, and a WBINVD is performed to write back and invalidate the caches, the processor may hang.

IMPLICATION: This region corresponds to a 256-Kbyte range beginning at 1792 Mbytes. Systems with cacheable memory in this region (i.e., with more than 1.792 Gbytes of DRAM) may hang while running software which performs WBINVD instructions. WBINVD is a privileged instruction only executable by operating systems

and driver software; some operating systems are known to execute this instruction, though infrequently (usually during bootup).

WORKAROUND: Using the variable MTRRs, a write-through (WT) memory region can be set up for the address range 70000000h-7003FFFFh in systems which support over 1.792 Gbytes of DRAM (this should be expanded to a 4-Mbyte region instead of 256 Kbytes, however, due to Erratum 29). This will prevent addresses in this range from being in the Modified state, thus preventing this erratum. However, this also may have a performance impact on the operating system; BIOS should modify the Int 15 E820h function to mark this range as "reserved" in the memory descriptor list provided to the OS through this function. Operating systems which do not use this function call may experience some performance degradation when accessing memory in this region (since it is not WB type). Also note that the WT memory type can be used to override a portion of a WB memory region, similar to UC memory type as currently documented; this will be changed in the specification (see Specification Changes).

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

51. UC Store May Be Reordered Around WC Transaction

PROBLEM: Store operations to uncacheable (UC) memory are intended to be system serializing events on the processor bus, with respect to transactions to write-combining (WC) memory. This means that all WC transactions before a subsequent UC store should complete before the store is allowed to execute, and any subsequent WC transactions should begin after the store completes. However, if the last cache line (or chunk) of data in a WC transaction is retried on the processor bus, a subsequent UC store may be reordered so that the store completes before the retried transaction. It is also possible for the transactions to be reordered in the processor core (as opposed to the external bus) and to be issued on the bus in reverse order. Note that UC loads operate as designed, and will never be reordered in this way.

IMPLICATION: If a device which uses WC memory is dependent on the completion of WC transactions before a UC store is executed, perhaps due to some designed-in effect of the UC store (such as activating a graphics accelerator sequence), this erratum may occur and the data from the last portion of the WC transaction would not be present, possibly causing the failure of the designed-in effect. Note that for a linear frame buffer in a video implementation, the effect of this erratum would be that some number of pixels would be updated on the screen on the order of a microsecond later than expected.

WORKAROUND: If system serialization by a UC store for WC memory is required, one of two workarounds may be used:

1. Insert a UC load before the UC store instruction. The WC transaction will not be reordered behind the UC transactions in this case.
2. Insert an *XCHG reg, stack* instruction before the UC store instruction. This will also guarantee the desired ordering.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

52. Spurious IFU Error Reported With MCEs Enabled

PROBLEM: The Pentium Pro processor can signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism in the event that the Instruction Fetch Unit (IFU) detects a difference in the consistency of, or the absence of code in the streaming buffers, victim cache or the instruction cache, i.e., a loss of inclusion. A loss of inclusion can occur in normal code execution. This condition requires the processor to detect the internal restarting of a memory load operation coincident with the internal signaling of a branch misprediction. If, with MCEs enabled, these two internal events occur and the speculative execution path of the mispredicted code encountered multiple L1 code cache hits to the same set as that from which the original load was executed, then a MCE may be signaled. With MCEs enabled, the possible loss of inclusion in the IFU is

detected and signaled prematurely, as such the unrecoverable MCE and operating system shutdown resulting from this errata, are essentially spurious.

IMPLICATION: If the conditions above are met and MCEs are enabled in the operating system, a spurious MCE may occur, causing operating system shutdown. If MCEs are disabled normal code execution with no interruption will result. However if MCEs are disabled and if self- or cross-modifying code is being executed, then unpredictable application behavior is possible, although current validation has shown execution to continue normally.

WORKAROUND: Enabling MCEs, as Intel recommends, will provide the system user with a higher confidence level in correct processor and code execution, but will allow the possibility of a spurious MCE in the above circumstances. Intel has not identified a workaround to prevent the spurious MCEs, when they are enabled.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

53. Extended TRDY# May Cause Hang With DBSY# Asserted

PROBLEM: The Pentium Pro processor asserts DBSY# during a transfer of data due to an implicit writeback for a write request which receives a HITM# (hit Modified) snoop response. If a responding agent asserts TRDY# for more than one clock beyond DBSY# deassertion (typically, TRDY# is deasserted the clock after DBSY# assertion is observed) to allow previously requested write data to transfer, the Pentium Pro processor may reassert DBSY# for the implicit writeback prematurely. Intel has only observed this erratum in a focused testing environment, when memory aliasing is used (i.e., in an MP system with processors mapping memory to different types, or when a memory type is changed dynamically by programming the Memory Type Range Registers (MTRRs) or changing a page-table entry's PCD bit on the fly).

IMPLICATION: This protocol violation may result in a system hang in systems with agents which assert TRDY# past the typical window.

WORKAROUND: Maintain a consistent MTRR memory model between processors in an MP system, and do not reprogram the MTRRs or change the PCD bits for main memory on the fly.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

54. Call Gates Can Cause Incorrect Handling of Stack Size Attribute

PROBLEM: Transitioning processor privilege levels via a call gate descriptor may result in the use of an ESP value with the top 16-bits cleared to 0's for the parameter copy portion of the call gate, if the CALL opcode is restarted. If no parameters are copied to the new stack, this erratum will not occur.

The opcode may be restarted due to a processor-initiated update of a page's Access bit, or due to the occurrence of a particular processor-initiated memory ordering event during the opcode's execution. A processor-initiated update of the Access bit occurs when a call gate requires a Task State Segment (TSS) whose page is marked as not having been accessed, or when the page containing a more privileged stack segment is accessed by the call gate. The processor-initiated memory ordering event occurs whenever an address compare (on address signals A#[19:5]) of a previous store from a different processor or DMA agent on the external bus matches the address of a page containing the TSS (or the kernel stack segment) being loaded by the call gate.

Using the incorrect value derived from the ESP during a call gate results in an unexpected page fault (Trap 0Eh) during the parameter copy, unless the miscalculated linear address for the caller's stack resolves to a page that is also present, and is accessible by the caller. If the page is present and accessible, it is possible that code could be executed using a stack containing incorrect data. At this point, system behavior is dependent upon what the called routine does given the incorrect stack data.

IMPLICATION: An application (or the operating system) may be terminated with a Page Fault Exception (Trap 0Eh) during a call gate, due to a miscalculated linear address for the caller's stack. In the event that the miscalculated linear address for the caller's stack resolves to a page that is also present and is accessible by the caller, system behavior becomes dependent upon the called routine's behavior when given the bad stack pointer.

WORKAROUND: For software using call gates, there are three possible complete workarounds for this erratum, any of which may be used alone or in combination:

1. Do not pass parameters using the parameter passing feature of a call gate,
2. Use a 32-bit stack in the routine or kernel being called, and either never clear the Access bit of the page containing the TSS or ensure that the Access bit is set when a call gate is performed, or
3. Align the TSS (or the page containing the TSS) on a 7 mod 32 linear address (i.e., 7, 39, 71, etc.), and either never clear the Access bits of the page containing the TSS or ensure that the Access bit is set when a call gate is performed.

In systems using sB1 silicon, BIOS code can contain a workaround for this erratum which precludes the need for any of the above three workarounds.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

55. LBER May Be Corrupted After Some Events

PROBLEM: The last branch record (LBR) and the last branch before exception record (LBER) can be used to determine the source and destination information for previous branches or exceptions. The LBR contains the source and destination addresses for the last branch or exception, and the LBER contains similar information for the last branch taken before the last exception. This information is typically used to determine the location of a branch which leads to execution of code which causes an exception. However, after a catastrophic bus condition which results in an assertion of BINIT# and the reinitialization of the buses, the value in the LBER may be corrupted. Also, after either a CALL which results in a fault or a software interrupt, the LBER and LBR will be updated to the same value, when the LBER should not have been updated.

IMPLICATION: The LBER and LBR registers are used only for debugging purposes. When this erratum occurs, the LBER will not contain reliable address information. The value of LBER should be used with caution when debugging branching code; if the values in the LBR and LBER are the same, then the LBER value is incorrect. Also, the value in the LBER should not be relied upon after a BINIT# event.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

56. SMI During REP String Instruction Clears CR4

PROBLEM: If an assertion of SMI# occurs during a REP string instruction, the system will enter System Management Mode (SMM). When this happens, CR4 is cleared and all information in this register is lost. This register contains information on system settings, such as the Page Global Enable (PGE) and Performance Counter Enable (PCE) bits.

IMPLICATION: The settings that are stored in the CR4 register are lost when the system enters SMM during a REP string instruction. This may cause an operating system or application to operate incorrectly or halt unexpectedly.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

57. BTMs May Be Corrupted During Simultaneous L1 Cache Line Replacement

PROBLEM: When Branch Trace Messages (BTMs) are enabled and such a message is generated, the BTM may be corrupted when issued to the bus by the L1 cache, if a new line of data is brought into the L1 data cache simultaneously. Though the new line being stored in the L1 cache is stored correctly, and no corruption occurs in the data, the information in the BTM may be incorrect due to the internal collision of the data line and the BTM.

IMPLICATION: Although BTM's may not be entirely reliable due to the erratum, the conditions for this erratum to occur have only been exhibited during focused simulation testing. Intel has not observed this erratum in a system level validation environment.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

58. Spurious Machine Check Exception Via IFU Data Parity Error

PROBLEM: The Pentium Pro processor can signal an unrecoverable Machine Check Exception (MCE) in the event that the Instruction Fetch Unit (IFU) detects a mismatch when verifying instruction parity. The execution of code which modifies the current instruction sequence that may already be fetched into the processor can cause an instruction at a given address to appear differently depending on when it was fetched in time relative to its being modified. Thus, a speculatively prefetched instruction may have been modified such that it now differs from the copy of the same instruction resident in the instruction cache. This discrepancy (of one copy located in the speculative prefetch portion, and a different copy in the instruction cache) is sensed by the IFU. When the IFU detects that the instruction stream has been modified, it flushes the pipeline and attempts to restart the instruction stream. In the interim, the IFU recognizes the disparate instructions described above, and signals a data parity error. The data parity error is signaled as an MCE before the instruction stream has had a chance to restart. This MCE will cause an operating system that has enabled MCE to shutdown. No incorrect code is executed by the processor in this situation (even if MCE is disabled). Note that this erratum occurs under a specific set of address dependencies and timing events.

IMPLICATION: Executing such a sequence of modifying code without proper synchronization may not always result in predictable program behavior. The processor's signaling of an MCE due to a data parity error in the IFU may then result in an unexpected system halt if the above conditions are met and MCEs are enabled.

WORKAROUND: For the sB1 stepping of the processor, disabling Instruction Streaming Buffers by setting bit 30 of the Model Specific Register (MSR) at address 33h to "1", can workaround this erratum. The effect on system performance of setting this bit has been found to be smaller than run-to-run measurement variations of conventional benchmarks such as SPECint* 95 or SYSmark* 32, but the effect may vary depending on workload. For prior steppings, it is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

59. Loss of Inclusion In IFU Can Cause Machine Check Exception

PROBLEM: The Pentium Pro processor can signal an unrecoverable Machine Check Exception (MCE) as a consistency checking mechanism in the event that the Instruction Fetch Unit (IFU) detects differences in the consistency of code in instruction streaming buffers against code resident in the instruction cache, i.e., a loss of inclusion. When application code makes an operating system call, the processor transitions execution privilege levels. If the code for the OS call is not already resident in the level 1 cache, then the processor may prefetch code while identifying a cache line(s) for eventual eviction to make space for the new code. Upon return from the OS call, the processor continues execution of application code at the user level. The processor, due to deep speculation and branch prediction, may attempt to execute instructions from the previously prefetched kernel

code starting by attempting to replace the victim line with kernel code in a buffer internal to the IFU. The IFU detects that the current application is insufficiently privileged to execute the kernel code and so, suppresses the eviction of the previously selected victim line. Despite having detected this condition, the IFU does replace this victim line with the kernel line. If the processor now attempts to restart execution of the current application code by refetching the original victim line it no longer finds it in the instruction cache. The IFU detects this loss of inclusion, and signals this by generating a MCE. If MCEs are enabled, this event can cause an operating system to shutdown. Note that this erratum occurs under a specific set of address dependencies and timing events.

IMPLICATION: The occurrence of all the conditions above can lead the IFU to signal a loss of inclusion by generating an MCE. If MCEs are enabled in the system, then the operating system may shutdown upon noticing the MCE resulting in system failure. If MCEs are disabled, then unpredictable application behavior is theoretically possible, although current validation has shown execution to continue normally.

WORKAROUND: For the sB1 stepping of the processor, disabling Instruction Streaming Buffers by setting bit 30 of the Model Specific Register (MSR) at address 33h to "1", can workaround this erratum. The effect on system performance of setting this bit has been found to be smaller than run-to-run measurement variations of conventional benchmarks such as SPECint 95 or SYSmark 32, but the effect may vary depending on workload. For prior steppings, it is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

60. TAP Controller Not Automatically Reset At Power-Up

PROBLEM: In the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*, in Chapter 10, "Pentium® Pro Processor Test Access Port," Section 10.2, page 10-3, under the Test-Logic-Reset bullet, in connection with the TAP controller reset, it is stated that: "The controller also enters this state immediately when TRST# is pulled active, and automatically upon power-up of the Pentium Pro processor." The sB1 stepping of the processor does not automatically reset the TAP upon power-up.

IMPLICATION: The TAP controller can also be reset by two additional means, which are documented in the workaround section below. If the TAP controller is not reset by one of these means, then it is possible for the sB1 stepping of the Pentium Pro processor to power-up in an indeterminate state preventing normal functionality.

WORKAROUND: The TAP controller logic can be reset by the assertion of TRST# (e.g., via pull-down) or the assertion of the TMS pin for 5 TAP clocks (TCK). Systems which pull TRST# low via a pull-down resistor are not affected by this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

61. Erroneous Signaling of User Mode Protection Violation

PROBLEM: If the Pentium Pro processor attempts to access a page in physical memory marked not present (Present bit clear), a page fault exception (#PF) is generated. Before proceeding, there is a narrow internal timing window where the processor verifies that no other higher priority fault conditions are present. During this time, it is possible for another agent to allocate a new page directory or page table entry (PDE/PTE) corresponding to the same linear address of the original access, writing new values into the PDE/PTE with the Access bit (A-bit) or the Dirty bit (D-bit) cleared. When the original processor completes its checking for other fault conditions, and re-examines the A/D bit of the recently modified PDE/PTE, it finds that it has been cleared. Internal hardware correctly signals this scenario as a condition to which the processor should respond by setting the A/D bit, but erroneously reports it as a generic paging protection violation. Instead of attempting to set the appropriate A/D bit, this event is reported as an Int14 with exception code 0x05, i.e., user mode protection violation.

IMPLICATION: The occurrence of this scenario will result in the erroneous signaling of a user mode protection violation instead of a page fault and may result in application termination depending on operating system

behavior in response to a user mode protection violation. Intel has only observed this erratum to date in laboratory testing of multi-processor systems.

WORKAROUND: Operating systems which allocate new PTEs and PDEs should set the Access bit (A-bit) and Dirty bit (D-bit) to work around this erratum. Alternately, an operating system's Int14 handler can determine if a protection violation condition truly exists, and if none is found, return without further action.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

62. Invalid Operation Not Signaled By The FIST Instruction on Some Out of Range Operands

PROBLEM: On certain, large, negative, floating-point operands, and only in three of the four possible processor rounding modes, the instructions FIST[P] m16int and FIST[P] m32int do not detect that the operand is so large that it will not fit into the target data size. As a consequence, the expected Invalid Operation exception response for this situation is not correctly provided, nor is the Invalid Operation flag set in the Floating Point status word as specified in the *Intel Architecture Programmers Reference Manual, Volume 2*. Under the failing conditions, noted below, the precision exception (#PE) flag will also be incorrectly set. The erratum occurs only when all of the following conditions are met:

1. The FIST[P] instruction is either a 16- or 32-bit operation; 64-bit operations are unaffected.
2. Either the '*to nearest* ', '*to zero*' or '*up*' rounding modes are being used. The round '*down*' mode is unaffected by this erratum.
3. The sign bit of the floating-point operand is negative.
4. The floating-point operand being converted is significantly more negative than can be described by the integer size being targeted.

ACTUAL vs. EXPECTED RESPONSE

A. Actual Response:

When the required conditions are encountered, the processor provides the following response:

- Return the MAXNEG value (8000h for FIST16 & 80000000h for FIST32) to memory.
- The IE (Invalid Operation) bit in the Floating Point status word is *not set* to flag the use of an invalid operand.
- The PE (precision error) bit the Floating Point status word is *set*.
- No exception handler is invoked.
- In the case of a FISTP instruction the Operand will have been popped from the floating point stack.

B. Expected Response

The expected processor response when the invalid operation exception is *masked* is:

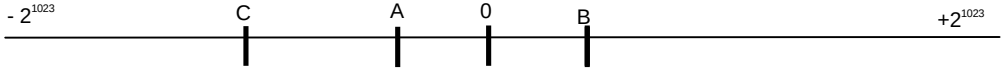
- Return the MAXNEG value (8000h for FIST16 & 80000000h for FIST32) to memory.
- The IE (Invalid Operation) bit in the Floating Point status word is *set* to flag the overflow.
- The PE (precision error) bit the Floating Point status word is *not set*.

The expected processor response when the invalid operation exception is *unmasked* is:

- Do not return a result to memory. Keep the original operand intact on the stack.
- The IE (Invalid Operation) bit in the Floating Point status word is *set* to flag the overflow.

- The PE (precision error) bit the Floating Point status word is *not set*.
- Vector to the user numeric exception handler.

IMPLICATION: Erroneous operation results when the operand is so large that it will not fit into the target data size. The operands affected by this erratum are significantly outside (by a factor of 3X) the range that can be, correctly, converted to an integer value. The figure below and corresponding table identifies the normal range of integer numbers (between A and B) and the starting point of the operands affected by this erratum. Discrete failing operands will be present in the range between point C and the maximum negative number that can be represented by the processor (-2^{1023} in double precision format). Note 2 below gives a qualitative description of the nature of the discrete failing values. Software that does not rely on the Invalid Operation exception flag being set and signaled by either an exception OR by software polling is not impacted by this erratum.



Range of potentially affected numbers.

Not all number in this range are affected

Range for a valid Integer

16-bit Operation	A	B	C
	-32,768.0	+32,767.0	< -98304.0
32-bit Operation	A	B	C
	-2,147,483,648.0	+2,147,483,647.0	< -6,442,450,944.0

WORKAROUND: Any of two software workarounds will avoid occurrence of this erratum:

Range checking performed prior to execution of the FIST[P] instruction will prevent the overflow condition from occurring, and may already be implemented as a standard coding style.

Software can use the presence of MAXNEG in the result integer to indicate that an out of range conversion may have occurred.

Note 1: A possible alternative is to use the FIST64 instruction to store the converted operand to memory and access the lower 16 or 32 bits as the required integer. Even though this mechanism will not signal an attempted out of range conversion with a 16 bit or 32 bit target, it is currently in use by many compilers today.

Note 2: The values affected by this erratum are those which contain an exponent value within the affected range, AND a specific bit pattern at a specific offset within the mantissa, AND at least one nonzero bit to the right of the above bit pattern. The offset within the mantissa is a function of the floating point exponent value. The specific bit pattern is 0x8000 for FIST16 and 0x80000000 for FIST32. This means that for any given exponent within the range, one mantissa value in every 216 possible mantissa values exhibits the erratum for FIST16, and one mantissa value in every 232 possible mantissa values exhibits the erratum for FIST32.

Examples of affected values for FIST16, 80 bit binary notation (not an exhaustive list)

(xx means any bit pattern, yy means any nonzero bit pattern)

sign	exponent	mantissa
1	100000000010100	1xxxxx1000000000000000yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000010101	1xxxxxx1000000000000000yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000010110	1xxxxxxx1000000000000000yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy

Examples of affected values for FIST32, 80 bit binary notation (not an exhaustive list)

(xx means any bit pattern, yy means any nonzero bit pattern)

sign	exponent	mantissa
1	100000000100011	1xxxx10000000000000000000000000000000yyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000100100	1xxxxx10000000000000000000000000000000yyyyyyyyyyyyyyyyyyyyyyyyy
1	100000000100101	1xxxxxx10000000000000000000000000000000yyyyyyyyyyyyyyyyyyyyyyyyy

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

63. EFLAGS May Be Incorrect After a Multiprocessor TLB Shootdown

PROBLEM: When the Pentium Pro processor executes a read-modify-write arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes. In this case the EFLAGS value pushed onto the stack of the page fault handler may be reflective of the status of the EFLAGS register after the instruction would have completed execution rather than that before it has executed under a certain set of circumstances. This class of instruction will initially perform a load operation that has the side effect of ensuring that the final store portion of the instruction will successfully complete. The load ensures this by verifying that the page table information of the page containing the data is in the DTLB. This page entry could be evicted from the DTLB by speculative loads from other instructions that hit the same way of the DTLB, before the store is executed. If in the very small window of time between the change of state of the page table and the shootdown becoming effective (i.e., the processor being told to remove the translation from the TLB), the page table entry has had its page permissions tightened (e.g., from Present to Not Present, or from Read/Write to Read Only, etc.) by the operating system in main memory by another processor (with no corresponding synchronization and subsequent TLB flush) the store will generate a DTLB miss and a call to the OS's page fault handler. The EFLAGS register may have already been updated by the arithmetic portion of the instruction before entry to the page fault handler. If under these circumstances the fault handler elects to restart the instruction, then the re-execution may generate an incorrect result. Instructions affected by this erratum are the memory destination forms of ADC, SBB, RCR & RCL (instructions that use a flag, carry, as input to the instruction). It should be noted that the locked version of these instructions is not impacted by this erratum.

IMPLICATION: This scenario can only occur in a multiprocessor system running under an operating system that implements a 'lazy' TLB shootdown. Lazy TLB shootdown occurs when one processor makes changes to the page tables, in memory, and then signals other processors to remove the page entry from their TLB without a multiprocessor synchronization being performed. To date, Intel has not observed this erratum in any laboratory testing of commercially available software applications.

In a multiprocessor system the arithmetic flags of the EFLAGS register and its memory stack image, may contain incorrect data if the read-modify-write arithmetic instruction encounters a page fault. Page Fault handler software that uses the resulting EFLAGS may see incorrect information. If the original instruction is restarted by

the page fault handler, the instruction may produce incorrect results based on the prior modifications of the EFLAGS register.

WORKAROUND: Software may use the locked form of the ADC, SBB, RCR & RCL instructions to avoid this erratum. Operating systems should ensure that no processor is currently accessing a page that is scheduled to have its page permissions tightened, e.g., moved from Present to Not Present or have a page fault handler that can handle any incorrect state. Intel is working with Multiprocessor Operating System vendors to ensure that an OS level workaround is implemented as required.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

64. Possible System Hang When Paging is Disabled and Re-enabled from Uncached Memory

PROBLEM: If paging is disabled via the PG bit of CR0 and then later re-enabled while executing code from a page marked uncachable by its Page Table Entry (PCD=1) but located in memory mapped as Write Back or Write Through by the processor MTRRs, the processor could internally enter a state resulting in a system hang.

IMPLICATION: Operating systems that enable and disable paging with the above described memory configurations could hang. This erratum has not been seen in any commercially available operating systems or applications.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

65. FLUSH# Assertion Disables L2 Machine Check Exception Reporting

PROBLEM: Upon FLUSH# assertion, the L2 Machine Check Exception generation is disabled. Once the FLUSH# pin is asserted, the processor disables the L2 MCA, by clearing the associated MCi_CTL control register to “0”’s. This operation is invisible to the software being executed.

IMPLICATION: Errors that should be reported by the L2 MCA are not reported from the time that the FLUSH# signal is asserted until the time that the MCi_CTL register is written back to all “1”s. All other errors will continue to be logged as normal.

WORKAROUND: Platform specific code (e.g., BIOS or system management software) has the potential for driving a device to assert the FLUSH# pin. If the platform specific code asserts the FLUSH# pin, this code should be enhanced to detect that MCA Exceptions are globally enabled (via register CR4.MCE). The code should then write “0”s to all of the MCi_CTL registers to clear any spurious entries and then write “1”s to all of the MCi_CTL registers in order to re-enable exception reporting. Hardware devices in systems that require L2 error reporting which could assert the FLUSH# pin should not assert FLUSH#.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section

66. Delayed Line Invalidation Issue During 2-Way MP Data Ownership Transfer

PROBLEM: In 2-way MP systems, each processor may attempt to modify a different portion of the same cache line, referenced as line ‘A’ in the discussion below. When this erratum occurs (with the following example given for a 2-way MP system with processors noted as ‘P0’ and ‘P1’), each processor contains a shared copy of line A

in both their L1 and L2 caches. Each processor must issue an invalidation cycle before that processor can definitively source the results of its internal write to a portion of line A to the other processors.

There exists a narrow timing window when, if P0 wins the external bus invalidation race and gains ownership rights to line A due to the sequence of bus invalidation traffic, P1 may not have completed the pending invalidation of its own, currently valid and shared copy of line A. During this window, it is possible for a P1 internal opportunistic write to a portion of line A (while awaiting ownership rights) to occur with the original shared copy of line A still resident in P1's L2 cache. Such internal modification is permissible subject to delaying the broadcast of such changes until line ownership has actually been gained. However, the processor must ensure that any internal re-read by P1 of line A returns with data in the order actually written; in this case, this should be the data written by P0. In the case of this erratum, the internal re-read uses the data which was written by P1.

IMPLICATION: Multiprocessor or threaded application synchronization that is implemented via operating system-provided synchronization constructs are not affected by this erratum. Applications which rely upon the usage of locked semaphores rather than memory ordering are also unaffected. Uniprocessor systems are not affected by this erratum. Intel has not identified, to date, any commercially available application or operating system software which is affected by this erratum. If the erratum does occur, the delayed line invalidation that occurs naturally due to the fact that one processor will necessarily win the invalidation race allows a narrow timing window to exist where one processor may re-read a line that it just wrote internally, but return with the stale data that was present from the previous shared state rather than the data written more recently by another processor.

WORKAROUND: Deterministic barriers beyond which program variables will not be modified can be achieved via the usage of locked semaphore operations, and this scheme has been shown to effectively work around this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

67. Potential Early Deassertion of LOCK# During Split-Lock Cycles

PROBLEM: During a split-lock cycle there are four bus transactions; 1st ADS# (a partial read), 2nd ADS# (a partial read), 3rd ADS# (a partial write), and the 4th ADS# (a partial write). Due to this erratum, LOCK# may deassert a clock after the 4th ADS# of the split-lock cycle instead of after the 4th RS# has been sampled. The following sequence of events are required for this erratum to occur:

1. A lock cycle occurs (split or nonsplit).
2. Five more bus transactions (assertion of ADS#) occur.
3. A split-lock cycle occurs and BNR# toggles after the 3rd ADS# (partial write) of the split-lock cycle. This in turn delays the assertion of the 4th ADS# of the split-lock cycle. BNR# toggling at this time could most likely happen when the bus is set for an IOQ depth of 2.

When all of these events occur, LOCK# will be deasserted in the next clock after the 4th ADS# of the split-lock cycle.

IMPLICATION: This may affect chipset logic which monitors the behavior of LOCK# deassertion.

WORKAROUND: None identified.

STATUS: For the steppings affected, see the Summary Table of Changes at the beginning of this section.

68. A20M# May Be Inverted After Returning From SMM and Reset

PROBLEM: This erratum is seen when software causes the following events to occur:

1. The assertion of A20M# in real address mode.
2. After entering the 1-Mbyte address wrap-around mode caused by the assertion of A20M#, there is an assertion of SMI# intended to cause a Reset or remove power to the processor. Once in the SMM handler, software saves the SMM state save map to an area of nonvolatile memory from which it can be restored at some point in the future. Then software asserts RESET# or removes power to the processor.
3. After exiting Reset or completion of power-on, software asserts SMI# again. Once in the SMM handler, it then retrieves the old SMM state save map which was saved in event 2 above and copies it into the current SMM state save map. Software then asserts A20M# and executes the RSM instruction. After exiting the SMM handler, the polarity of A20M# is inverted.

IMPLICATION: If this erratum occurs, A20M# will behave with a polarity opposite from what is expected (i.e., the 1-Mbyte address wrap-around mode is enabled when A20M# is deasserted, and does not occur when A20M# is asserted).

WORKAROUND: Software should save the A20M# signal state in nonvolatile memory before an assertion of RESET# or a power down condition. After coming out of Reset or at power on, SMI# should be asserted again. During the restoration of the old SMM state save map described in event 3 above, the entire map should be restored, except for bit 5 of the byte at offset 7F18h. This bit should retain the value assigned to it when the SMM state save map was created in event 3. The SMM handler should then restore the original value of the A20M# signal.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

69. EFLAGS Discrepancy on a Page Fault After a Multiprocessor TLB Shootdown

PROBLEM: This erratum may occur when the Pentium Pro processor executes one of the following read-modify-write arithmetic instructions and a page fault occurs during the store of the memory operand: ADD, AND, BTC, BTR, BTS, CMPXCHG, DEC, INC, NEG, NOT, OR, ROL/ROR, SAL/SAR/SHL/SHR, SHLD, SHRD, SUB, XOR, and XADD. In this case, the EFLAGS value pushed onto the stack of the page fault handler may reflect the status of the register after the instruction would have completed execution rather than before it. The following conditions are required for the store to generate a page fault and call the operating system page fault handler:

1. The store address entry must be evicted from the DTLB by speculative loads from other instructions that hit the same way of the DTLB before the store has completed. DTLB eviction requires at least three load operations that have linear address bits 15:12 equal to each other and address bits 31:16 different from each other in close physical proximity to the arithmetic operation.
2. The page table entry for the store address must have its permissions tightened during the very small window of time between the DTLB eviction and execution of the store. Examples of page permission tightening include from Present to Not Present or from Read/Write to Read Only, etc.
3. Another processor, without corresponding synchronization and TLB flush, must cause the permission change.

IMPLICATION: This scenario may only occur on a multiprocessor platform running an operating system that performs “lazy” TLB shootdowns. The memory image of the EFLAGS register on the page fault handler’s stack prematurely contains the final arithmetic flag values although the instruction has not yet completed. Intel has not identified any operating systems that inspect the arithmetic portion of the EFLAGS register during a page fault nor observed this erratum in laboratory testing of software applications.

WORKAROUND: No workaround is needed upon normal restart of the instruction, since this erratum is transparent to the faulting code and results in correct instruction behavior. Operating systems may ensure that no processor is currently accessing a page that is scheduled to have its page permissions tightened or have a page fault handler that ignores any incorrect state.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

70. Near CALL to ESP Creates Unexpected EIP Address

PROBLEM: As documented, the CALL instruction saves procedure linking information in the procedure stack and jumps to the called procedure specified with the destination (target) operand. The target operand specifies the address of the first instruction in the called procedure. This operand can be an immediate value, a general purpose register, or a memory location. When accessing an absolute address indirectly using the stack pointer (ESP) as a base register, the base value used is the value in the ESP register before the instruction executes. However, when accessing an absolute address directly using ESP as the base register, the base value used is the value of ESP *after* the return value is pushed on the stack, not the value in the ESP register *before* the instruction executed.

IMPLICATION: Due to this erratum, the processor may transfer control to an unintended address. Results are unpredictable, depending on the particular application, and can range from no effect to the unexpected termination of the application due to an exception. Intel has observed this erratum only in a focused testing environment. Intel has not observed any commercially available operating system, application, or compiler that makes use of or generates this instruction.

WORKAROUND: If the other seven general purpose registers are unavailable for use, and it is necessary to do a CALL via the ESP register, first push ESP onto the stack, then perform an *indirect* call using ESP (e.g., CALL [ESP]). The saved version of ESP should be popped off the stack after the call returns.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

71. Memory Type Undefined for Nonmemory Operations

PROBLEM: Memory Type for nonmemory operations such as I/O and Special cycles are not driven as the uncacheable (UC) memory type. Although the Memory Type Attribute for nonmemory operations usually manifests itself as UC, this feature is not designed into the implementation, and the memory type during nonmemory cycles is undefined.

IMPLICATION: Bus agents could decode a non-UC memory type for nonmemory operations.

WORKAROUND: Bus agents must consider transaction type to determine the validity of the Memory Type Attribute.

STATUS: For the steppings affected, see the Summary Table of Changes at the beginning of this section.

72. Bus Protocol Conflict With Optimized Chipsets

PROBLEM: A “dead” turnaround cycle with no agent driving the address, request command, or request parity signals must occur between the processor driving these signals and the chipset driving them after asserting BPRI#. The Pentium Pro processor does not follow this protocol. Thus, if a system uses a chipset or third party agent which optimizes its arbitration latency (reducing it to 2 clocks when it observes an active (low) ADS# signal and an inactive (high) LOCK# signal on the same clock that BPRI# is asserted (driven low)), the Pentium Pro processor may cause bus contention during an unlocked bus exchange.

IMPLICATION: This violation of the reduced arbitration latency bus exchange protocol may cause a system-level setup timing violation on the address, request command, or request parity signals on the system bus. This may result in a system hang or assertion of the AERR# signal, causing spurious corrective action or shutdown of the system, as the system hardware and software dictate. The possibility of failure due to the contention caused by this erratum may be increased due to the processor’s internal active pull-up of these signals on the clock after the signals are no longer being driven by the processor.

WORKAROUND: If the chipset and third party agents used with the Pentium Pro processor do not optimize their arbitration latency as described above, no action is required. If agents that have implemented this optimization are used, decreasing the system bus frequency may reduce the possibility of a failure caused by the contention described above.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

73. FP Data Operand Pointer May Not Be Zero After Power On or Reset

PROBLEM: The FP Data Operand Pointer, as specified, should be reset to zero upon power on or Reset by the processor. Due to this erratum, the FP Data Operand Pointer may be nonzero after power on or Reset.

IMPLICATION: Software which uses the FP Data Operand Pointer and count on its value being zero after power on or Reset without first executing an FINIT/FNINIT instruction will use an incorrect value, resulting on incorrect behavior of the software.

WORKAROUND: Software should follow the recommendation in Section 8.2 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (Order Number 243192). This recommendation states that if the FPU will be used, software-initialization code should execute an FINIT/FNINIT instruction following a hardware reset. This will correctly clear the FP Data Operand Pointer to zero.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

74. Premature Execution of a Load Operation Prior to Exception Handler Invocation

PROBLEM: This erratum can occur with any of the following situations:

1. if an instruction that performs a memory load causes a code segment limit violation
2. if a waiting floating-point instruction

With any of the above circumstances, it is possible that the load portion of the instruction will have prematurely executed before the exception handler is entered.

IMPLICATION: In normal code execution where the target of the load operation is to write back memory there is no impact from the load being prematurely executed, nor from the restart and subsequent re-execution of that instruction by the exception handler. If the target of the load is to uncached memory that has a system side effect, restarting the instruction may cause unexpected system behavior.

WORKAROUND: Code that loads from memory that has side effects can effectively workaround this behavior by using simple integer based load instructions when accessing side effect memory and by ensuring that all code is written such that a code segment limit violation cannot occur as a part of reading to side effect memory.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

75. Read Portion of RMW Instruction May Execute Twice

PROBLEM: When the Pentium Pro processor executes a read-modify-write arithmetic instruction, with memory as the destination, it is possible for a page fault to occur during the execution of the store on the memory operand after the read operation has completed but before the write operation completes. If the memory targeted for the instruction is UC (uncached), memory will observe the occurrence of the initial load before the page fault handler and again if the instruction is restarted.

IMPLICATION: If the memory targeted for the RMW instruction has no side effects, then the memory location will simply be read twice with no additional implications. If, however, the load targets a memory region that has side effects, multiple occurrences of the initial load may lead to unpredictable system behavior.

WORKAROUND: IHV's and ISV's who write device drivers for custom hardware that may have a side effect style of design should use simple loads and simple stores to transfer data to and from the device.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

76. Intervening Writeback May Occur During Locked Transaction

PROBLEM: During a transaction which has the LOCK# signal asserted (i.e., a locked transaction), there is a potential for an explicit writeback caused by a previous transaction to complete while the bus is locked. The explicit writeback will only be issued by the processor which has locked the bus, and the lock signal will not be deasserted until the locked transaction completes, but the atomicity of a lock may be compromised by this erratum. Note that the explicit writeback is an expected cycle, and no memory ordering violations will occur. This erratum is, however, a violation of the bus lock protocol.

IMPLICATION: A chipset or third-party agent (TPA) which tracks bus transactions in such a way that locked transactions may only consist of a read-write or read-read-write-write locked sequence, with no transactions intervening, may lose synchronization of state due to the intervening explicit writeback. Systems using chipsets or TPAs which can accept the intervening transaction will not be affected.

WORKAROUND: The bus tracking logic of all devices on the system bus should allow for the occurrence of an intervening transaction during a locked transaction.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

77. MC2_STATUS MSR Has Model-Specific Error Code and Machine Check Architecture Error Code Reversed

PROBLEM: The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* documents that for the MC*i*_STATUS MSR, bits 15:0 contain the MCA (machine-check architecture) error code field and bits 31:16 contain the model-specific error code field. However, for the MC2_STATUS MSR, these bits have been reversed. For the MC2_STATUS MSR, bits 15:0 contain the model-specific error code field and bits 31:16 contain the MCA error code field.

IMPLICATION: A machine check error may be decoded incorrectly if this erratum on the MC2_STATUS MSR is not taken into account.

WORKAROUND: When decoding the MC2_STATUS MSR, reverse the two error fields

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

78. MOV With Debug Register Causes Debug Exception

PROBLEM: When in V86 mode, if a MOV instruction is executed on debug registers, a general-protection exception (#GP) should be generated, as documented in the *Pentium® Pro Family Developer's Manual, Volume 3: System Programming Guide*, Section 10.2, "Debug Registers," and the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Section 14.2, "Debug Registers." However, in the case when the general detect enable flag (GD) bit is set, the observed behavior is that a debug exception (#DB) is generated instead.

IMPLICATION: With debug-register protection enabled (i.e., the GD bit set),and attempting to execute a MOV on debug registers in V86 mode, a debug exception will be generated instead of the expected general-protection fault.

WORKAROUND: In general, operating systems do not set the GD bit when they are in V86 mode. The GD bit is generally set and used by debuggers. The debug exception handler should check that the exception did not occur in V86 mode before continuing. If the exception did occur in V86 mode, the exception may be directed to the general-protection exception handler.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

79. Data Breakpoint Exception in a Displacement Relative Near Call May Corrupt EIP

PROBLEM: If a data breakpoint is programmed at the memory location where the stack push of a near call is performed, the processor will update the stack and ESP appropriately, but may skip the code at the destination of the call. Hence, program execution will continue with the next instruction immediately following the call, instead of the target of the call.

IMPLICATION: The failure mechanism for this erratum is that the call would not be taken; therefore, instructions in the called subroutine would not be executed. As a result, any code relying on the execution of the subroutine will behave unpredictably.

WORKAROUND: Do not program a data breakpoint exception on the stack where the push for the near call is performed.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

80. Misprediction in Program Flow May Cause Unexpected Instruction Execution

PROBLEM: To optimize performance through dynamic execution technology, the P6 architecture has the ability to predict program flow. In the event of a misprediction, the processor will normally clear the incorrect prediction, adjust the EIP to the correct location, and flush out any instructions it may have fetched from the misprediction. In circumstances where a branch misprediction occurs, the correct target of the branch has already been opportunistically fetched into the streaming buffers, and the L2 cycle caused by the evicted cache line is retried by the L2 cache, the processor may fail to flush out the retirement unit before the speculative program flow is committed to a permanent state.

IMPLICATION: The results of this erratum may range from no effect to unpredictable application or OS failure. Manifestations of this failure may result in:

- Unexpected values in EIP,
- Faults or traps (e.g., page faults) on instructions that do not normally cause faults,
- Faults in the middle of instructions, or
- Unexplained values in registers/memory at the correct EIP.

WORKAROUND: It is possible for BIOS code to contain a workaround for this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

81. RDMSR and WRMSR To Invalid MSR Address May Not Cause GP Fault

PROBLEM: The RDMSR and WRMSR instructions allow reading or writing of MSRs (Model Specific Registers) based on the index number placed in ECX. The processor should reject access to any reserved or unimplemented MSRs by generating #GP(0). However, there are some invalid MSR addresses for which the processor will not generate #GP(0).

IMPLICATION: For RDMSR, undefined values will be read into EDX:EAX. For WRMSR, undefined processor behavior may result.

WORKAROUND: Do not use invalid MSR addresses with RDMSR or WRMSR.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

82. SYSENTER/SYSEXIT Instructions Can Implicitly Load "Null Segment Selector" to SS and CS Registers

PROBLEM: According to the processor specification, attempting to load a null segment selector into the CS and SS segment registers should generate a General Protection Fault (#GP). Although loading a null segment selector to the other segment registers is allowed, the processor will generate an exception when the segment register holding a null selector is used to access memory.

However, the SYSENTER instruction can implicitly load a null value to the SS segment selector. This can occur if the value in SYSENTER_CS_MSR is between FFF8h and FFFBh when the SYSENTER instruction is executed. This behavior is part of the SYSENTER/SYSEXIT instruction definition; the content of the SYSTEM_CS_MSR is always incremented by 8 before it is loaded into the SS. This operation will set the null bit in the segment selector if a null result is generated, but it does not generate a #GP on the SYSENTER instruction itself. An exception will be generated as expected when the SS register is used to access memory, however.

The SYSEXIT instruction will also exhibit this behavior for both CS and SS when executed with the value in SYSENTER_CS_MSR between FFF0h and FFF3h, or between FFE8h and FFEBh, inclusive.

IMPLICATION: These instructions are intended for operating system use. If this erratum occurs (and the OS does not ensure that the processor never has a null segment selector in the SS or CS segment registers), the processor's behavior may become unpredictable, possibly resulting in system failure.

WORKAROUND: Do not initialize the SYSTEM_CS_MSR with the values between FFF8h and FFFBh, FFF0h and FFF3h, or FFE8h and FFEBh before executing SYSENTER or SYSEXIT.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

83. PRELOAD Followed by EXTEST Does Not Load Boundary Scan Data

PROBLEM: According to the IEEE 1149.1 Standard, the EXTEST instruction would use data "typically loaded onto the latched parallel outputs of boundary-scan shift-register stages using the SAMPLE/PRELOAD instruction prior to the selection of the EXTEST instruction." As a result of this erratum, this method cannot be used to load the data onto the outputs.

IMPLICATION: Using the PRELOAD instruction prior to the EXTEST instruction will not produce expected data after the completion of EXTEST.

WORKAROUND: None identified at this time.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

84. Far Jump to New TSS With D-bit Cleared May Cause System Hang

PROBLEM: A task switch may be performed by executing a far jump through a task gate or to a new Task State Segment (TSS) directly. Normally, when such a jump to a new TSS occurs, the D-bit (which indicates that the page referenced by a Page Table Entry (PTE) has been modified) for the PTE which maps the location of the previous TSS will already be set and the processor will operate as expected. However, if the D-bit is clear at the time of the jump to the new TSS, the processor will hang.

IMPLICATION: If an OS is used which can clear the D-bit for system pages, and which jumps to a new TSS on a task switch, then a condition may occur which results in a system hang. Intel has not identified any commercial software which may encounter this condition; this erratum was discovered in a focused testing environment.

WORKAROUND: Ensure that OS code does not clear the D-bit for system pages (including any pages that contain a task gate or TSS). Use task gates rather than jumping to a new TSS when performing a task switch.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

1AP. APIC Access to Cacheable Memory Causes SHUTDOWN

PROBLEM: APIC operations which access memory with any type other than uncacheable (UC) is illegal, and if machine check exceptions (MCEs) are disabled, the CPU will enter shutdown after such an access. If MCEs are enabled, an MCE will occur. However, in this circumstance, a second MCE will be signaled. This will also cause the Pentium Pro processor to enter shutdown.

IMPLICATION: Recovery from a PIC access to cacheable memory will not be successful. Correctly written software will not encounter this erratum.

WORKAROUND: Ensure that the memory space to which PIC accesses can be made is marked as type UC (uncacheable) in the memory type range registers (MTRRs) to avoid this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

2AP. Possible Hang Due to Catastrophic Errors During BSP Determination

PROBLEM: A catastrophic error during the bootstrap processor (BSP) determination process should cause the assertion of IERR#. If the catastrophic error is due to the APIC data bus being stuck at electrical zero, then the system hangs without asserting IERR#.

IMPLICATION: Systems may hang during boot due to a catastrophic error. Intel has not observed this erratum under normal system operation; it was found in a test environment using a grounded APIC data bus.

WORKAROUND: None identified.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

3AP. INIT_IPI After STARTUP_IPI-STARTUP_IPI Sequence May Cause AP to Execute at 0h

PROBLEM: The MP Specification states that to wake up an application processor (AP), the interprocessor interrupt sequence INIT_IPI, STARTUP_IPI, STARTUP_IPI should be sent to that processor. On the Pentium Pro processor, an INIT_IPI, STARTUP_IPI sequence will also work. However, if the INIT_IPI, STARTUP_IPI, STARTUP_IPI sequence is sent to an AP, an internal race condition may occur in the APIC logic which leaves the processor in an incorrect state. Operation will be correct in this state, but if another INIT_IPI is sent to the processor, the processor will not stop execution as expected, and will instead begin execution at linear address 0h. In order for the race condition to cause this incorrect state, the system's core to bus clock ratio must be 5:2 or greater.

IMPLICATION: If a system is using a core to bus clock ratio of 5:2 or greater, and the sequence INIT_IPI, STARTUP_IPI, STARTUP_IPI is generated on the APIC bus to wake up an AP, and then at some later time another INIT_IPI is sent to the processor, that processor may attempt to execute at linear address 0h, and will execute random opcodes. Some operating systems do generate this sequence when attempting to shut the system down, and in a multiprocessor system, may hang after taking the processors offline. The effect seen will be that the OS may not restart the system if 'shutdown and restart' or the equivalent is selected upon exiting the operating system. If an operating system gives the user the capability to take an AP offline using an INIT_IPI (Intel has not identified any operating systems which currently have this capability), this option should not be used.

WORKAROUND: BIOS code should execute a single STARTUP_IPI to wake up an application processor. Operating systems, however, will issue an INIT_IPI, STARTUP_IPI, STARTUP_IPI sequence, as recommended in the MP specification. It is possible that BIOS code may contain a workaround for this erratum in systems with C0 or subsequent steppings of Pentium Pro processor silicon. No workaround is available for the B0 stepping of the Pentium Pro processor.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

4AP. Small Window for System Hang After INIT# or INIT_IPI

PROBLEM: There is a very small timing window during which assertion of the INIT# signal or reception of an INIT_IPI may cause a processor to hang, due to an internal race condition. This window is dependent on the relationship of the APIC clock to the processor BCLK.

IMPLICATION: The system may hang upon receiving an INIT_IPI or assertion of the INIT# signal. This has only been observed by Intel in a diagnostic/test environment where the INIT# signal was asserted very frequently. If the system hangs due to this erratum, power cycling or performing a hard reset will restart the system successfully.

WORKAROUND: Though Intel has never observed this failure in a system which uses a synchronous APIC clock, it is still theoretically possible to encounter this erratum in such a configuration. Using an APIC clock frequency ratio of greater than 4.5 (i.e., BCLK / APIC clock frequency ≥ 4.5) will avoid this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

5AP. Virtual Wire Mode Through Local APIC May Cause Int 15

PROBLEM: If the APIC subsystem is configured in mixed mode with Virtual Wire mode implemented through the local APIC, an interrupt vector of 0Fh (Intel reserved encoding) may be generated by the local APIC (Int 15). This vector may be generated upon receipt of a spurious interrupt (an interrupt which is removed before the system receives the INTA sequence) instead of the programmed 8259 spurious interrupt vector.

IMPLICATION: The spurious interrupt vector programmed in the 8259 is normally handled by an operating system's spurious interrupt handler. However, a vector of 0Fh is unknown to some operating systems, which would crash if this erratum occurred.

WORKAROUND: BIOS should disable the local APIC in uniprocessor systems and MP systems which do not use an I/O APIC, by setting bit 11 in the APICBASE MSR at address 1Bh to '0'. In MP systems with an I/O APIC, BIOS should configure the local APIC in Virtual Wire mode through the I/O APIC. In the I/O APIC's redirection table, LVT0 should be programmed for EXTINT interrupts; all other LVT entries should be masked. The MP Specification Table should be modified to reflect this configuration, as well. Symmetric I/O mode may also be used; this mode will not encounter this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

6AP. Divide By Zero Error on LINT0 Glitch

PROBLEM: If the APIC subsystem is configured in mixed mode with Virtual Wire mode implemented through the local APIC, an interrupt vector of 0 (Divide by Zero exception) may be generated by the local APIC (Int 0). This vector may be generated when the LINT0 signal to the local APIC is asserted, then deasserted and rapidly reasserted again. If the reassertion coincides with the receipt of the corresponding Interrupt Acknowledge (INTA) cycle from the processor core by the local APIC, a vector of 00h is returned by the local APIC to the processor core. The processor will then generate a Divide by Zero exception.

IMPLICATION: If this occurs, the resulting Int 0 may cause an operating system to crash with a Divide by Zero error message.

WORKAROUND: BIOS should disable the local APIC in uniprocessor systems and MP systems which do not use an I/O APIC, by setting bit 11 in the APICBASE MSR at address 1Bh to '0'. In MP systems with an I/O APIC, BIOS should configure the local APIC in Virtual Wire mode through the I/O APIC. In the I/O APIC's redirection table, LVT0 should be programmed for EXTINT interrupts; all other LVT entries should be masked. The MP Specification Table should be modified to reflect this configuration, as well. Symmetric I/O mode may also be used; this mode will not encounter this erratum.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

7AP. Write to Mask LVT (Programmed as EXTINT) Will Not Deassert Outstanding Interrupt

PROBLEM: If the APIC subsystem is configured in Virtual Wire Mode implemented through the local APIC, (i.e., the LINT0 signal is connected to the 8259A and LVT1 is programmed as EXTINT), a write that sets the mask bit of the LVT will not deassert an outstanding interrupt if the LINT0 signal is already asserted. The interrupt will be erroneously posted to the processor core despite the attempt to mask the LVT.

IMPLICATION: Because of the masking attempt, interrupts may be generated from an unexpected source when the system expects no interrupts to be posted.

WORKAROUND: Software can issue a write to the IMR of the 8259A to mask all interrupt sources, which will cause the deassertion of the LINT0 signal. This should be followed by a read to the 8259A IRR to ensure that the LINT0 signal has been deasserted. Software may then issue the write to mask LVT entry 1.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

8AP. APs Do Not Respond to a STARTUP_IPI After an INIT# or INIT_IPI in Low Power Mode

PROBLEM: When an INIT_IPI is received, all application processors (APs) wait for a STARTUP_IPI message. The AP starts executing at a location derived from this message, once it is received. While the AP is waiting for a STARTUP_IPI, the core clock is stopped in order to save on power if the Low Power feature is enabled via bit 26 of the EBL_POWERON MSR. The core clock is not re-enabled when a STARTUP_IPI is received, preventing the AP from restarting.

IMPLICATION: If Low Power Mode is enabled, the APs will not restart when a STARTUP_IPI message is received after an INIT# or INIT_IPI. Note that uniprocessor systems are not affected by this erratum.

WORKAROUND: Do not enable Low Power Mode in MP systems.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

9AP. Serial Bus Interrupts May Be Lost in APIC Mixed Mode

PROBLEM: If LINT0 or LINT1, set as EXTINT, are asserted, the processor internally issues an Interrupt Acknowledge (INTA) cycle. If the asserted interrupt is then deasserted, as a spurious interrupt, before the internal INTA cycle has completed, the INTA cycle is nonetheless sent to the external system bus. However, if a serial interrupt is received just as the processor has decided to send the INTA from spurious interrupt to the external bus, the spurious interrupt bus cycle is completed correctly but the serial bus interrupt cycle is never completed.

IMPLICATION: The APIC will block the service of any equal or lower priority serial interrupts until an internal INTA cycle is received for the initial serial bus interrupt possibly leading to a system hang. In a system where only the serial bus is used or only virtual wire through the local or I/O APIC is used this errata will not be seen.

WORKAROUND: Avoid using the APIC in mixed mode, which has the potential of losing serial interrupts due to this erratum. Use the APIC in either virtual wire through the local APIC, virtual wire through the I/O APIC or Symmetric APIC mode only.

STATUS: For the steppings affected see the Summary Table of Changes at the beginning of this section.

DOCUMENTATION CHANGES

The Documentation Changes listed in this section apply to the *Pentium® Pro Family Developer's Manual*, *Volumes 1, 2, and 3*. All Documentation Changes will be incorporated into a future version of the appropriate Pentium Pro processor documentation.

1. PE Bit Number in FPU Status Word, Volume 2, Page 7-50

Section 7.8.6, "Inexact-Result (Precision) Exception (#P)," paragraph 2 of this section should read "The inexact-result exception flag (PE) is bit **5** of the FPU status word, and the mask bit (PM) is bit **5** of the FPU control word."

2. GTL+ Edge Rate Range Incorrect

Figures 12-4 and 12-14 in the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications,* document falling edge rate ranges in the waveforms for characterizing receiver ringback tolerance and setup time. Both figures document ranges of 0.3 V/ns to 3.0 V/ns; these values should be 0.3 V/ns to **1.5 V/ns**. Also, Figure 12-13 documents the rising edge rate range as 0.3 V/ns to 1.5 V/ns; this should be 0.3 V/ns to **0.8 V/ns**.

3. Power-on Configuration Register Documented Inconsistently

In the *Pentium® Pro Family Developer's Manual*, the Power-on Configuration Register (or EBL_CR_POWERON) values documented in Section 9.3 of *Volume 1: Specifications,* and Appendix C of *Volume 3: Operating System Writer's Guide*, are inconsistent. The correct values are presented below.

Register Address			
Hex	**Dec**	**Register Name**	**Bit Description**
2AH	42	EBL_CR_POWERON	
		0	Reserved*
		1	Data Error Checking Enable 1 = Enabled 0 = Disabled Read/Write
		2	Response Error Checking Enable FRCERR Observation Enable 1 = Enabled 0 = Disabled Read/Write
		3	AERR# Driver Enable 1 = Enabled 0 = Disabled Read/Write
		4	BERR# Driver Enable for Initiator Bus Requests 1 = Enabled 0 = Disabled Read/Write
		5	Reserved

Register Address			
Hex	**Dec**	**Register Name**	**Bit Description**
		6	BERR# Driver Enable for Initiator Internal Errors 1 = Enabled 0 = Disable Read/Write
		7	BINIT# Driver Enable 1 = Enabled 0 = Disabled Read/Write
		8	Output Tristate Enabled 1 = Enabled 0 = Disabled Read
		9	Execute BIST 1 = Enabled 0 = Disabled Read
		10	AERR# Observation Enabled 1 = Enabled 0 = Disabled Read
		11	Reserved
		12	BINIT# Observation Enabled 1 = Enabled 0 = Disabled Read
		13	In Order Queue Depth 1 = 1 0 = 8 Read
		14	1 Mbyte Power-on Reset Vector 1 = 1 Mbyte 0 = 4 Gbytes Read
		15	FRC Mode Enable = Enabled 0 = Disabled Read
		17:16	APIC Cluster ID Read
		19:18	Reserved
		21:20	Symmetric Arbitration ID Read
		24:22	Clock Frequency Ratio Read

Register Address			
Hex	**Dec**	**Register Name**	**Bit Description**
		25	Reserved
		26	Low Power Enable ead/Write
		63:27	Reserved

*Bit 0 of this register has been redefined several times, and is no longer used in the Pentium® Pro processor.

4. GTL+ Minimum Overshoot Specification

The minimum overshoot specified in Table 11-12 in the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications,* is incorrectly stated as 0.55 mV. This value (T# α in Figure 11-10) should be **100** mV, as specified in the *Pentium® Pro Processor at 150 MHz, 166 MHz, 180 MHz, and 200 MHz* datasheet.

5. Conditional Move Opcode Incorrect

In Section 11.3 of the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, the description of the opcode for *CMOVcc—Conditional Move* should be 0F 4*x* /r, rather than 0F 4*x* cw/cd.

6. 32 Entries for 4-Kbyte Pages in ITLB

In Table 11-1, "Characteristics of the Caches, TLBs, and Write Buffer," in the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*, the instruction TLB is listed as having 64 entries for 4-Kbyte pages. It should list the instruction TLB as having 32 entries for 4-Kbyte pages, 4-way set associative.

7. A5# Value Should be H for Arb Id 1

In Table 9-3, "Arbitration ID Configuration," in the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*, lists an A5# value of L for Arb Id 1. This should be an H.

8. Register Names Incorrect for CPUID Return Value

In Table 11-7, "Information Returned by CPUID Instruction," in the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, the initial EAX value of 0 is shown to return "inel" into the ECX register and "ntel" into the EDX register. These names are reversed; the "inel" is returned into EDX, and the "ntel" is returned into ECX, as described in the text below the table.

9. *Corrections to* Volume 2: Programmer's Reference Manual

A number of typos and other documentation errors will be corrected in the next revision of the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual.* A list of significant changes is given below. Note that other changes may be made, and not all significant changes may be listed here.

- Page 2-6: There are two Dispatch/Execute Units depicted in Figure 2-1. The lower of the two should be listed as an **Instruction Pool (Reorder Buffer)**.

- Page 6-16: In Figure 6-1, “Operation of the PUSH Instruction,” the position of the doubleword value after pushing is one doubleword higher than depicted (at **n - 4**, not at n - 8).
- Page 6-17: The third and fifth sentences of the last paragraph reference the address-size attribute of the stack, and should reference the **operand**-size attribute.
- Page 6-29: The last sentence of Section 6.9.1.3, “Return-From-Interrupt Instruction,” is incorrect and will be deleted.
- Page 11-69: Under “Flags Affected,” the ZF flag is set if the values in the destination operand and register AL, AX, or EAX are **equal**.
- Page 11-82: The example for the DAA instruction is incorrect, and should read:

```
ADD AL, BL  Before: AL=79H  BL=35H  EFLAGS(0SZAPC)=XXXXXX
            After:  AL=AEH  BL=35H  EFLAGS(0SZAPC)=110000
DAA         Before: AL=2EH  BL=35H  EFLAGS(0SZAPC)=110000
            After:  AL=04H  BL=35H  EFLAGS(0SZAPC)=X00101
```

- Page 11-101: The first entry in the table under the ST(0) header should be **-∞**.
- Page 11-104: The last sentence of the description for the FCHS instruction should read “The following table shows the results obtained when **changing the sign** of various classes of numbers.”
- Page 11-122: The first entry of the table under the +0 header should be **-∞**. Under the +∞ header, the entries for a SRC of +I or +F should be **+0**, not +∞.
- Page 11-143: The description of the FMULP instruction is incorrect; the result is actually stored in **ST(1)**.
- Page 11-183: The description of the FSUBRP ST(i), ST(0) instruction is incorrect; it subtracts **ST(*i*)** from **ST(0)**, stores the result in ST(*i*), and pops the register stack.
- Page 11-184: The row and column headers for the table should be interchanged.
- Page 11-211: The operation of the INC instruction is DEST ← DEST **+** 1.
- Page 11-233: The TASK-RETURN parameters are (* PE=1, VM=**0**, NT=1 *).
- Page 11-241: Near and far jumps are sometimes referred to as intrasegment and intersegment **jumps**, not calls.
- Page 11-243: The last first-level IF statement on this page should start IF far **jump**...
- Page 11-250/276: On these two pages, the tables give Type 6 the name “16-bit trap gate” and Type 7 the name “16-bit interrupt gate.” These names are reversed. This also applies to Types E and F.
- Page 11-292: For the MOVZX instruction, the second paragraph of the description should be deleted. The first paragraph should state that the instruction **zero** extends the value.
- Page 11-308: The second paragraph of the description should begin “The current **operand**-size attribute...”
- Page 11-328: For the RDMSR instruction, a #GP fault will not be generated if the current privilege level is not 0 in the Real Address Mode. The first condition under Real Address Mode Exceptions should be deleted.

10. Remapping WB, WT Memory Types

On page 11-21 of the *Pentium® Pro Family Developer’s Manual, Volume 3: Operating System Writer’s Guide*, Section 11.11.6, “Remapping Memory Types,” rule 2 states that the write-through type cannot be mapped into the write-back type. This is incorrect; the statement should state that the write-back type cannot be mapped into the write-through type.

11. BTM "From" and "To" Fields Reversed

In Section 5.2.3.5 ("Branch Trace Message") of the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*, the information provided on the data lines is documented in reverse order. The document should specify that D[63:32]# contain either the address of the first byte of the branch instruction or the address of the instruction immediately following the branch, and D[31:0]# contain the linear address of the target.

12. Numeric Underflow Exception Description Clarification

Chapter 7 of the *Pentium® Pro Family Developer's Manual*, *Volume 2*: *Programmer's Reference Manual,* Section 7.8.5 on pages 7-50 and 7-51 describes the Numeric Underflow Exception. There is a typographical error in the first paragraph of this description.

The second line of the first paragraph on page 7-51 reads: "Here, if overflow occurs again, after the result has been biased, a properly signaled 0 is stored in the destination operand." This sentence should read "Here, if *underflow* occurs again, after the result has been biased, a properly signaled 0 is stored in the destination operand."

13. Page Directory Pointer Table Entry Present Bit Identification

In the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*, Chapter 3, "Protected-Mode Memory Management," contains Figures 3-20 and 3-21 on pages 3-31 and 3-32 respectively, which are visual representations of the bit definitions for the Page Directory Pointer Table entries. Bit 0 of this entry should be labeled "P" signifying a "Present Bit," and needs to be set to '1' anytime Extended Physical Addressing mode is used.

14. Flushing Caches Upon Entering SMM

Section 9.4.2 on page 9-6 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*, describes SMRAM Caching. The first paragraph suggests flushing the processor's caches upon entering SMM by executing an INVD instruction—this should read "by executing a WBINVD instruction."

15. General Protection Fault Description

Table 5-1 on page 5-3 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*, tabulates Protected Mode Exceptions and Interrupts. Entry 13 suggests the interrupt or exception type for General Protection (#GP) to be Fault/Trap—the correct type is only a Fault, not a Trap.

16. Invalid Arithmetic Operations And Masked Responses To Them Relative to FIST/FISTP Instruction

The *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, on page 7-47 shows Table 7-20 and the *Intel Architecture Software Developer's Manual, Volume 1*, Table 7-20 shows "Invalid Arithmetic Operations and the Masked Responses to Them." The table entry corresponding to the FIST/FISTP condition is missing, and is shown below:

Condition	Masked Response
FIST/FISTP Instruction when input operand <> MAXINT for destination operand size.	Return MAXNEG to destination operand.

17. Limit Checking and G Flag Usage Correction

The *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual,* in Section 4.3, "Limit Checking," on page 4-4 describes the usage of the G flag with respect to limit checking. The second sentence of the second paragraph is missing a '^' character, and should read: "When the G flag is set (4-Kbyte page granularity), the processor scales the value in the limit field by a factor of 2^12."

18. MCI_ADDR MSR Reference Section Correction

The first sentence of Section 16.3.2.3 on page 16-6 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual,* contains a reference to a previous section, but incorrectly identifies the referenced section number. The first sentence should read: "The MCi_ADDR MSR contains the address of the code or data memory location that produced the machine-check error if the ADDRV flag in the MCi_STATUS register is set (see Section 16.3.2.2, "MCi_STATUS MSR")."

19. MC2_STATUS Bit Description Correction

Table C-1 in Appendix C of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual,* on page C-6 shows the bit description for the various fields of MCi_STATUS registers. For register at hexadecimal address 401H, i.e., "MC0_STATUS," MC_STATUS_MSCOD and MC_STATUS_MCACOD are shown to be represented by bit 31:16 and 15:0 respectively; these values are reversed in the documentation. MC_STATUS_MSCOD is contained is 15:0 and MC_STATUS_MCACOD is contained in 31:16.

20. FCOMI/FCOMIP/FUCOMI/FUCOMIP Setting of Flags Relative to Exceptions

Page 11-112 of the *Pentium® Pro Family Developer's Manual, Volume 2: Instruction Set Reference,* shows a table for FCOMI/FCOMIP/FUCOMI/FUCOMIP comparison results, where the last entry in the table "Unordered" has an asterisk (*) beside it referencing a table note that reads: "Note: * Flags not set if unmasked invalid-arithmetic operand (#IA) exception is generated; however this note should read: "Note: * Flags are set regardless, whether there is an unmasked invalid operand (#IA) exception generated or not.

21. Cache Management Instructions Correction

Section 11.6 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide,* will change as follows:

The INVD and WBINVD instructions are used to invalidate the contents of the L1 and L2 caches. The INVD instruction invalidates all internal (data and instruction) cache entries, then generates a special bus cycle (called a FLUSH cycle) that indicates that external caches (level 3 caches) also should be invalidated. The INVD instruction should be used with care. It does not force a write-back of modified cache lines; therefore, it can cause the caches to become inconsistent with system memory. Unless there is a specific requirement or benefit

to invalidating the caches without writing back the modified lines (such as, during testing or fault recovery where cache coherency with main memory is not a concern), software should use the WBINVD instruction.

The WBINVD instruction first writes back any modified lines in the caches, then invalidates the contents of both the L1 and L2 caches. It ensures that the cache coherency with main memory is maintained regardless of the write policy in effect (that is, write-through or write-back). Following this operation, the WBINVD instruction then generates a SYNC cycle, that directs the external caches to also write back modified data, and finally, a FLUSH cycle to invalidate external caches.

22. PUSH Does Not Pad With Zeros

The *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, page 4-2 and the *Intel Architecture Software Developer's Manual, Volume 1*, page 4-3, contain a section regarding stack alignment. The last sentence in the first paragraph of this section, which reads "If a 16-bit value is pushed onto a 32-bit wide stack, the value is automatically padded with zeros out to 32-bits." should be removed. The PUSH instruction does not pad with zeros.

23. DR7, Bit 10 is Reserved

The *Pentium® Pro Family Developer's Manual, Volume 3: Operating Systems Writer's Manual,* shows Figure 10-1, "Debug Registers." Bit 10 of DR7 should be "Reserved" instead of "1."

24. Additional States That Are Not Automatically Saved and Restored

In Section 9.4.1 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating Systems Writer's Manual*, the end of section lists the registers that are not automatically saved and restored following an SMI and the RSM instruction, respectively. The last two paragraphs should be as follows:

The following state is not automatically saved and restored following an SMI and the RSM instruction, respectively:

- Debug registers DR0 through DR3.
- The FPU registers.
- The MTRRs.
- Control register CR2.
- The model-specific registers (for the P6 family and Pentium® processors) or test registers TR3 through TR7 (for the Pentium and Intel486™ processors).
- The state of the trap controller.
- The machine-check architecture registers.
- The APIC internal interrupt state (ISR, IRR, etc.).
- The Microcode Update state.

If an SMI is used to power down the processor, a power-on reset will be required before returning to SMM, which will reset much of this state back to its default values. So an SMI handler that is going to trigger power down should first read these registers listed above directly, and save them (along with the rest of RAM) to nonvolatile storage. After the power-on reset, the continuation of the SMI handler should restore these values, along with the

rest of the system's state. Anytime the SMI handler changes these registers in the processor it must also save and restore them.

NOTE

A small subset of the MSRs (such as the time-stamp counter and performance-monitoring counter) are not arbitrarily writeable and therefore cannot be saved and restored. SMM-based power-down and restoration should only be performed with operating systems that do not use or rely on the values of these registers. Operating system developers should be aware of this fact and ensure that their operating-system assisted power-down and restoration software is immune to unexpected changes in these register values.

25. Cache and TLB Description Correction

In the Section 11.3 of the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, Table 11-10, and in Section 3.2 of the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, Table 3-7 (in the description of the CPUID instruction) the correct description for descriptor value 02H should be as follows:

Descriptor Value	Cache or TLB Description
02H	Instruction TLB: 4M-Byte Pages, **fully** associative, **2** entries

Also, the third bullet after the table should be as follows:

- Bytes 1, 2, and 3 of register EAX indicate that the processor contains the following:
  - 01H–A 32-entry instruction TLB (4-way set associative) for mapping 4-Kbytes pages.
  - 02H–A **2**-entry instruction TLB (**fully** associative) for mapping 4-Mbyte pages.
  - 03H–A 64-entry data TLB (4-way set associative) for mapping 4-Kbyte pages.

Finally, for the *Pentium® Pro Family Developer's Manual, Volume 3: Operating Systems Writer's Manual*, Table 11-1, the following corrections should be made:

Cache or Buffer	Characteristics
Instruction TLB (Large Pages)	**2** entries, **fully** associative

26. SMRAM State Save Map Contains Documentation Errors

In the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*, Chapter 9, "System Management Mode," Table 9-1 incorrectly documents the SMBASE+Offset for IDT Base and GDT Base on the Pentium Pro processor.

The storage locations for these parameters are model specific (i.e., they may differ between the Pentium processor, the Pentium Pro processor, and other P6 family proliferations). These entries in the tables above will be changed to Reserved. Hardware and software may not rely on the contents of these Reserved regions.

27. OF and DF of the EFLAGS Register are Mislabeled as System Flags

In the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual,* Table 3-7, "EFLAGS Register," and the *Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture,* Table 3-7, "EFLAGS Register," the Overflow Flag (OF) and Direction Flag (DF) are both incorrectly labeled as System Flags. The Overflow Flag should be labeled as a Status Flag and the Direction Flag should be labeled as a Control Flag.

28. CS:EIP Pushed Onto Stack Prior to Code Segment Limit Check

The *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, Section 11.3, and the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference*, Section 3.4, contain a detailed definition of the CALL instruction. In this definition, all instances where the instruction pointer is checked to ensure it is within the acceptable code segment limit followed by the CS:EIP register being pushed on the stack are in error. CS:EIP is pushed on the stack prior to the check of the instruction pointer. This means that in the case of a GP#(0) being generated due to an out-of-range instruction pointer, these values will be present on the stack.

29. BREQ# Sampled Incorrectly

In the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*, Section 4.1.5.1, "Reset Conditions," the third paragraph states, "The first BREQ# sample point is 2 clocks after RESET# is driven inactive." The first BREQ# sample point is actually one clock after RESET# is driven inactive. This reference and further implications of this documentation change should be taken into account when generating a reset condition.

30. Corrections to Opcode Maps

In Appendix A, "Opcode Map" in the *Pentium® Pro Family Developer's Manual Volume 3: Operating System Writer's Manual,* and in Appendix A, "Opcode Map" in the *Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference,* are the one and two byte opcode maps. The following tables are intended to replace those tables in their entirety:

Table A-1. One-Byte Opcode Map[1]

	0	1	2	3	4	5	6	7
0	ADD						PUSH	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	ES	ES
1	ADC						PUSH	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	SS	SS
2	AND							DAA
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	=ES	
3	XOR							AAA
	Eb,Gb	Ev,Gv	Gb,Eb	Gb,Ev	AL,Ib	eAX,Iv	=SS	
4	INC general register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
5	PUSH general register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSHA/ PUSHAD	POPA/ POPAD	BOUND	ARPL			Operand	Address
			Gv,Ma	Ew,Gw	=FS	=GS	Size	Size
7	Short-displacement jump on condition (Jb)							
	JO	JNO	JB/JNAE/ JC	JNB/ JAE/JNC	JZ/JE	JNZ/ JNE	JBE/ JNA	JNBE/ JA
8	Imm Group 1[2]			Imm Group 1[2]	TEST		XCHG	
	Eb,Ib	Ev,Iv	Ev,Ib	Eb,Ib	Eb,Gb	Ev,Gv	Eb,Gb	Ev,Gv
9	NOP	XCHG word or double-word register with eAX						
		eCX	eDX	eBX	eSP	eBP	eSI	eDI
A	MOV				MOVSB	MOVSW	CMPSB	CMPSW
	AL,Ob	eAX,Ov	Ob,AL	Ov,eAX	Xb,Yb	Xv,Yv	Xb,Yb	Xv,Yv
B	MOV immediate byte into byte register							
	AL	CL	DL	BL	AH	CH	DH	BH
C	Shift Group 2[2]		RET near		LES	LDS	MOV	
	Eb,Ib	Ev,Ib	Iw		Gv,Mp	Gv,Mp	Eb,Ib	Ev,Iv
D	Shift Group 2[2]				AAM	AAD		XLAT/ XLATB
	Eb,1	Ev,1	Eb,CL	Ev,CL				
E	LOOPNE/ LOOPNZ	LOOPE/ LOOPZ	LOOP	JCXZ/ JECXZ	IN		OUT	
	Jb	Jb	Jb	Jb	AL,Ib	eAX,Ib	Ib,AL	Ib,eAX

Table A-1. One-Byte Opcode Map[1] (Continued)

F	LOCK		REPNE	REP/ REPE	HLT	CMC	Unary Group 32	
							Eb	Ev
	8	9	A	B	C	D	E	F
0	OR						PUSH	2-byte
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	CS	Escape
1	SBB						PUSH	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	DS	DS
2	SUB							DAS
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv		
3	CMP							AAS
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	AL,Ib	eAX,Iv	=DS	
4	DEC General-Purpose Register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
5	POP Into General-Purpose Register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
6	PUSH	IMUL	PUSH	IMUL	INSB	INSW/D	OUTSB	OUTSW/ D
	Iv	Gv,Ev,Iv	Ib	Gv,Ev,Ib	Yb,DX	Yv,DX	Dx,Xb	DX,Xv
7	Short-Displacement Jump on Condition (Jb)							
	JS	JNS	JP/JPE	JNP/JPO	JL/JNGE	JNL/JGE	JLE/JNG	JNLE/JG
8	MOV					LEA	MOV	POP
	Eb,Gb	Ev,Gv	Gb,Eb	Gv,Ev	Ew,Sw	Gv,M	Sw,Ew	Ev
9	CBW	CWD/ CDQ	CALL	FWAIT	PUSHF/ PUSHFD	POPF/ POPFD	SAHF	LAHF
			Ap		Fv	Fv		
A	TEST		STOS/ STOSB	STOS/STO SW/STOTS D	LODSB	LODSW/L ODSD	SCAS/ SCACSB	SCASW/ SCASD/ SCAS
	AL,Ib	eAX,Iv	Yb,AL	Yv,eAX	AL,Xb	eAX,Xv	AL,Yb	eAX,Yv
B	MOV Immediate Word or Double Into Word or Double Register							
	eAX	eCX	eDX	eBX	eSP	eBP	eSI	eDI
C	ENTER	LEAVE	RET far	RET far	INT 3	INT	INTO	IRET
	Iw, Ib		Iw			Ib		
D	ESC (Escape to Coprocessor Instruction Set)							
E	CALL	JMP			IN		OUT	
	Jv	Jv	Ap	Jb	AL,DX	eAX,DX	DX,AL	DX,eAX
F	CLC	STC	CLI	STI	CLD	STD	Group 4[2]	Group 5[2]

NOTES:

1. All blanks in the opcode map are reserved and should not be used. Do not depend on the operation of these undefined opcodes.
2. Bits 5, 4, and 3 of ModR/M byte used as an opcode extension (see Section A.4).

Table A-2. Two Byte Opcode Map (First byte is 0FH)[1]

	0	1	2	3	4	5	6	7
0	Group 6[2]	Group 7[2]	LAR	LSL			CLTS	
			Gv,Ew	Gv,Ew				
1								
2	MOV							
	Rd,Cd	Rd,Cd	Cd,Rd	Dd,Rd				
3	WRMSR	RDTSC	RDMSR	RDPMC				
4	CMOVO	CMOVNO	CMOVB/ CMOVC/ CMOVNAE	CMOVAE/ CMOVNB/ CMOVNC	CMOVE/ CMOVZ	CMOVNE /CMOVN Z	CMOVBE /CMOVN A	CMOVA/ CMOVNB E
	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev
5								
6	PUNPCKLB W	PUNPCK LWD	PUNPCKLD Q	PACKSSD W	PCMPGT B	PCMPGT W	PCMPGT D	PACKUS WB
	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd	Pq, Qd
7	Group A[2]				PCMPEQ B	PCMPEQ W	PCMPEQ D	EMMS
		PSHIMW[3]	PSHIMD[3]	PSHIMQ[3]	Pq, Qd	Pq, Qd	Pq, Qd	
8	Long-Displacement Jump on Condition (Jv)							
	JO	JNO	JB/JNAE/ JC	JAE/JNB/JN C	JE/JZ	JNE/JNZ	JBE/JNA	JA/JNBE
9	Byte Set on condition (Eb)							
	SETO	SETNO	SETB/ SETC/ SETNA	SETAE/ SETNB/ SETNC	SETE/ SETG/ SETZ	SETNE/ SETNZ	SETBE/ SETNA	SETA/ SETNBE
A	PUSH	POP	CPUID	BT	SHLD	SHLD		
	FS	FS	Ev,Gv	Ev,Gv,Ib	Ev,Gv,CL			
B	CMPXCHG	CMPXCH G	LSS	BTR	LFS	LGS	MOVZX	
	Eb,Gb	Ev,Gv	Mp	Ev,Gv	Mp	Mp	Gv,Eb	Gv,Ew
C	XADD	XADD						Group 9[2]
	Eb,Gb	Ev,Gv						
D		PSRLW	PSRLD	PSRLQ		PMULLW		

Table A-2. Two Byte Opcode Map (First byte is 0FH)[1] (Continued)

		Pq, Qd	Pq, Qd	Pq, Qd		Pq, Qd		
E		PSRAW	PSRAD			PMULHW		
		Pq, Qd	Pq, Qd			Pq, Qd		
F		PSLLW	PSLLD	PSLLQ		PMADDWD		
		Pq, Qd	Pq, Qd	Pq, Qd		Pq, Qd		
	8	9	A	B	C	D	E	F
0	INVD	WBINVD		UD24				
1								
2								
3								
4	CMOVS	CMOVNS	CMOVP/ CMOVPE	CMOVNP/C MOVPO	CMOVL/ CMOVNGE	CMOVGE /CMOVNL	CMOVLE/ CMOVNG	CMOVG/ CMOVNLE
	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev	Gv, Ev
5								
6	PUNPCKHBW	PUNPCKHWD	PUNPCKHDQ	PACKSSDW			MOVD	MOVQ
	Pq,Qd	Pq,Qd	Pq,Qd	Pq,Qd			Pd,Ed	Pq,Qq
7							MOVD	MOVQ
							Ed,Pd	Qq,Pq
8	Long-Displacement Jump on Condition (Jv)							
	JS	JNS	JP/JPE	JNP/JPO	JL/JNGE	JNL/JGE	JLE/JNG	JNLE/JG
	Byte set on condition (Eb)							
9	SETS	SETNS	SETP/ SETPE	SETNP/ SETPO	SETL/ SETNGE	SETNL/ SETGE	SETLE/ SETNG	SETNLE
	Eb	Eb	Eb	Eb	Eb	Eb	Eb	Eb
A	PUSH	POP	RSM	BTS	SHRD	SHRD		IMUL
	GS	GS		Ev,Gv	Ev,Gv,Ib	Ev,Gv,CL		Gv,Ev
B		Invalid Opcode[4]	Group 8[2]	BTC	BSF	BSR	MOVSX	
			Ev,Ib	Ev,Gv	Gv,Ev	Gv,Ev	Gv,Eb	Gv,Ew
C	BSWAP							
	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI

Table A-2. Two Byte Opcode Map (First byte is 0FH)[1] (Continued)

D	PSUBUSB	PSUBUSW		PAND	PADDUSB	PADDUSW		PANDN
	Pq,Qq	Pq,Qq		Pq,Qq	Pq,Qq	Pq,Qq		Pq,Qq
E	PSUBSB	PSUBSW		POR	PADDSB	PADDSW		PXOR
	Pq,Qq	Pq,Qq		Pq,Qq	Pq,Qq	Pq,Qq		Pq,Qq
F	PSUBB	PSUBW	PSUBD		PADDB	PADDW	PADDD	
	Pq,Qq	Pq,Qq	Pq,Qq		Pq,Qq	Pq,Qq	Pq,Qq	

NOTES:

1. All blanks in the opcode map are reserved and should not be used. Do not depend on the operation of these undefined opcodes.
2. Bits 5, 4, and 3 of ModR/M byte used as an opcode extension (see Section A.4).
3. These abbreviations are not actual mnemonics. When shifting by immediate shift counts, the PSHIMD mnemonic represents the PSLLD, PSRAD, and PSRLD instructions, PSHIMW represents the PSLLW, PSRAW, and PSRLW instructions, and PSHIMQ represents the PSLLQ and PSRLQ instructions. The instructions that shift by immediate counts are differentiated by the ModR/M bytes (see Section A.4).
4. Use the 0F0B opcode (UD2 instruction) or the 0FB9H opcode when deliberately trying to generate an invalid opcode exception (#UD).

31. MP Initialization Protocol Algorithm Correction

In Section 7.5.6 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual,* and in Section 7.6.5 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, the algorithm for MP Initialization is defined. It is stated "the APIC hardware observes the BNR# (block next request) and BPRI# (priority agent bus request) pins to guarantee that the initial BIPI is not issued on the APIC bus until the BIST sequence is complete for all processors in the system." This is not correct. Only the observation of BNR# is required for the APIC hardware to proceed.

32. Interrupt 13-General Protection Exception (#GP)

In Section 5.12 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* and in Section 5.12 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*, a description of the exception interrupts is provided. In the description section of Interrupt 13-General protection Exception (#GP), the last bullet applies if the PAE and/or PSE flags are set, rather than just the PAE flag as reported in the documentation.

33. Addition to the LOCK Instruction Description

In the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, Section 11.2, "Instruction Set Reference," the description of the LOCK instruction is incomplete. The following paragraph should be added:

When the LOCK prefix is prefixed to an instruction and the memory area being accessed is write-back memory and is completely contained in the cache line in the processor, the LOCK# signal is generally not asserted. Instead, only the processor's cache is locked. Here, the processor's cache coherency mechanism insures that the operation is carried out atomically with regards to memory. See "Effects of a Locked Operation on Internal Processor Caches" in Chapter 7 of the *Pentium® Pro Family Developer's Manual*, *Volume 3,* for more information on locking of caches.

SPECIFICATION CLARIFICATIONS

The Specification Clarifications listed in this section apply to the *Pentium® Pro Family Developer's Manual*, *Volumes 1, 2, and 3,* and the *Pentium® Pro Processor with 1 MB L2 Cache at 200 MHz* datasheet. All Specification Clarifications will be incorporated into a future version of the appropriate Pentium Pro processor documentation.

1. Signal Edge Rate for 3.3 V Tolerant, APIC, and JTAG Signals

The following note will be added to the setup and hold times for the 3.3 V tolerant, APIC, and boundary scan signals (T12, T13, T28, T29, T43, and T44):

"Specified over the rise time (Tr) and fall time (Tf) ranges of 0.3 ns to 2 ns for these signals, between 0.8 V and 2.0 V (as defined by Figure 11-7)."

The title "Generic Clock Waveform" of Figure 11-7 will be changed to "Generic Waveform" to allow this reference.

2. OUTS Instruction and Subsequent Instructions

The description for the OUTS instruction in the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual*, 6th paragraph, contains incorrect information regarding sampling of the EWBE# pin, which is not a Pentium Pro processor signal. This paragraph should document that upon execution of an OUTS, OUTSB, OUTSW, or OUTSD instruction, the Pentium Pro processor will not execute the next instruction until the data phase of the transaction is complete.

3. Interrupt Recognition Determines Priority

The interrupt priority documented in Table 5-2 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*, reflects the order in which interrupts will be serviced upon simultaneous recognition by the processor (for example, when multiple interrupts are pending at an instruction boundary). This table does not necessarily reflect the order in which interrupts will be recognized by the processor if received simultaneously at the processor pins.

4. References to 2-Mbyte Pages Should Include 4-Mbyte Pages

Generically, "large pages" refers to either 2-Mbyte or 4-Mbyte pages. In Section 3.8 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*, 2-Mbyte pages are often referenced alone, when the behavior of 4-Mbyte pages is identical; these references should include all large pages.

5. Modification of Reserved Areas in the SMRAM Saved State Map

If data is incorrectly written to reserved areas of the saved state map, the processor will enter the shutdown state. This can also occur if invalid state information is saved in the SMRAM (such as if illegal combinations of bits are written to CR0 or CR4 before an SMI is serviced). CR4 is not distinctly part of the saved state map, as implied in Section 9.3.1.1 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*.

6. FRCERR Asserted for FRC Checker BIST Failure

In Section 3.4.8, in the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*, the last paragraph states: "When BIST completes, the Pentium Pro processor deasserts FRCERR if BIST succeeds and continues to assert FRCERR if BIST fails." This is only true for the FRC checker in an FRC system; in a non-FRC system, FRCERR is always deasserted, regardless of the BIST result. There is no external indication of BIST failure in a non-FRC system or for an FRC master; to determine the results of BIST, the result must be read from EAX after BIST (a nonzero result indicates failure).

7. Deassertion of BREQn# During RESET#

The description of BREQ*n*# deassertion protocol in Section 4.1.5.1, "Reset Conditions," in the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications,* does not match that described elsewhere in the document (including Table 11-14, "Reset Conditions A.C. Specifications"). This section will be modified as follows:

4.1.5.1 Reset Conditions

Table 11-14, "Reset Conditions A.C. Specifications," describes the timing requirements for configuration signals on observation of active RESET#. When a reinitialization condition is generated by the activation of BINIT#, BREQ*n*# must remain deasserted until 4 clocks after BINIT# is driven inactive. The first BREQ*n*# sample point is 4 clocks after BINIT# is sampled inactive.

8. TLB Flush Necessary After PDPE Change

As described in Section 3.7, "Translation Lookaside Buffers (TLBs)," in the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual,* the operating system is required to invalidate the corresponding entry in the TLB after any change to a page-directory or page-table entry. However, if the physical address extension (PAE) feature is enabled to use 36-bit addressing, a new table is added to the paging hierarchy, called the page directory pointer table (as per Section 3.8, "Physical Address Extension"). If an entry is changed in this table (to point to another page directory), the TLBs must then be flushed by writing to CR3.

9. LOCK# Deasserted Between Locked Sequences

The following clarification will be added to the description of the bus lock action in Sections 3.4.2, "Arbitration Phase Signals," and A.1.37, "LOCK# (I/O)," of the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications,* and to Section A.37, "LOCK# (I/O)," of the *Pentium® Pro Processor with 1 MB L2 Cache at 200 MHz* datasheet:

The LOCK# signal is always deasserted between two sequences of locked transactions on the Pentium Pro processor bus.

10. EXF4# Assertion in SMM

Section 3.4.3, "Request Signals," in the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*, lists the EXF[4:0]# signals and their functions in Table 3-11. The text for this table will contain the following clarification of the behavior of EXF4#:

EXF4# (SMM Memory) is asserted by the Pentium Pro processor during the second clock of the Request Phase in each transaction when the processor is in System Management Mode and indicates that the processor is accessing a separate "shadow" memory, the SMRAM.

11. Preventing Caching

Section 11.5.2 in the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide,* and Section 9.5.2 in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* document the procedure to prevent the L1 and L2 caches from performing all caching operations. However, this procedure differs from that given in Section 11.11.8 in the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide,* and Section 9.11.8 in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. The correct procedure that should be used is as follows:

1. Enter the no-fill cache mode. (Set the CD flag in control register CR0 to 1 and the NW flag to 0.)
2. Flush all caches using the WBINVD instruction.
3. Disable the MTRRs and set the default memory type to uncached, or set all MTRRs for the uncached memory type (see the discussion of the TYPE field and the E flag in Section 11.11.2.1, "MTRRdefType Register").

The caches must be flushed when the CD flag is cleared to insure system memory coherency. If the caches are not flushed in step 2, cache hits on reads will still occur and data will be read from valid cache lines.

12. MC*i*_CTL Registers Not Cleared by INIT#

During an assertion of the INIT# signal, all Pentium Pro processor bus agents are reset without affecting their internal caches or floating-point registers, as stated in Section 3.4.1 of the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*. This section should also state that the Machine Check Architecture registers (MC*i*_CTL) are also unaffected by the INIT# sequence.

13. Stack Use During Interrupts, Exceptions, and CALLs

Sections 4.3.6 and 4.4.1 of the *Pentium® Pro Family Developer's Manual, Volume 2: Programmer's Reference Manual,* contain inconsistencies which will be clarified to reflect the actual behavior of the architecture, as follows:

4.3.6 CALL and RET Operation Between Privilege Levels

When making a call to a more privileged protection level, the processor does the following (see Figure 4-3):

1. Performs an access rights check (privilege check).
2. Temporarily saves (internally) the current contents of the SS, ESP, CS, and EIP registers.
3. Loads the segment selector and stack pointer for the new stack (that is, the stack for the privilege level being called) from the TSS into the SS and ESP registers and switches to the new stack.
4. Pushes the temporarily saved SS and ESP values for the calling procedure's stack onto the new stack.
5. Copies the parameters from the calling procedure's stack to the new stack. (A value in the call gate descriptor determines how many parameters to copy to the new stack.)
6. Pushes the temporarily saved CS and EIP values for the calling procedure to the new stack.
7. Loads the segment selector for the new code segment and the new instruction pointer from the call gate into the CS and EIP registers, respectively.

Begins execution of the called procedure at the new privilege level.

When executing a return from the privileged procedure, the processor performs these actions:

1. Performs a privilege check.
2. Restores the CS and EIP registers to their values prior to the call.
3. (If the RET instruction has an optional n argument.) Increments the stack pointer by the number of bytes specified with the n operand to release parameters from the stack. If the call gate descriptor specifies that one or more parameters be copied from one stack to the other, a RET n instruction must be used to release the parameters from both stacks. Here, the n operand specifies the number of bytes occupied on each stack by the parameters. On a return, the processor increments ESP by n for each stack to step over (effectively remove) these parameters from the stacks.
4. Restores the SS and ESP registers to their values prior to the call, which causes a switch back to the stack of the calling procedure.
5. (If the RET instruction has an optional n argument.) Increments the stack pointer by the number of bytes specified with the n operand to release parameters from the stack (see explanation in step 3).
6. Resumes execution of the calling procedure.

See Chapter 4, “Protection,” in the *Pentium® Pro Family Developer's Manual, Volume 3,* for detailed information on calls to privileged levels and the call gate descriptor.

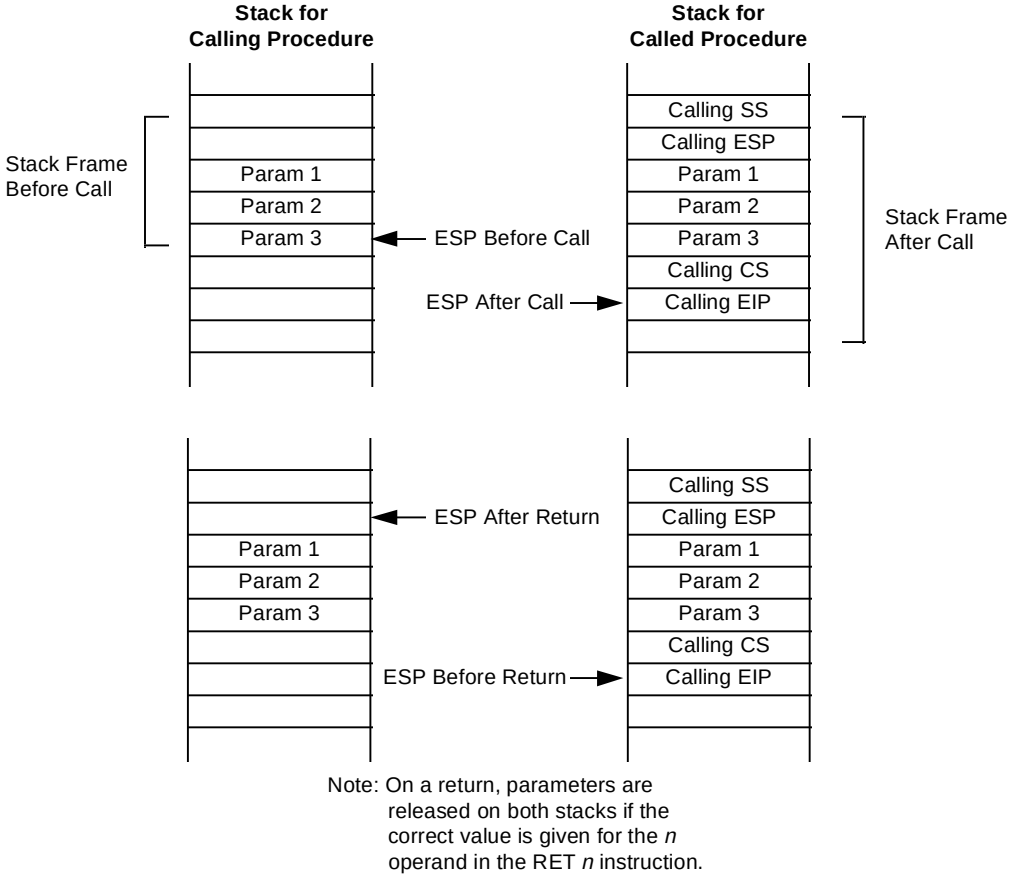


Figure 4-3. Stack Switch on a Call to a Different Privilege Level

4.4.1 Call and Return Operation for Interrupt or Exception Handling Procedures

A call to an interrupt or exception handler procedure is similar to a procedure call to another protection level (as described in Section 4.3.6, “CALL and RET Operation Between Privilege Levels”). Here, the interrupt vector references one of two kinds of gates: an interrupt gate or a trap gate. Interrupt and trap gates are similar to call gates in that they provide the following information:

- Access rights information.
- The segment selector for the code segment that contains the handler procedure.
- An offset into the code segment to the first instruction of the handler procedure.

The difference between an interrupt gate and a trap gate are as follows. If an interrupt or exception handler is called through an interrupt gate, the processor clears the interrupt enable (IF) flag in the EFLAGS register to prevent subsequent interrupts from interfering with the execution of the handler. When a handler is called through a trap gate, the state of the IF flag is not changed.

If the code segment for the handler procedure has the same privilege level as the currently executing program or task, the handler procedure uses the current stack; if the handler executes at a more privileged level, the processor switches to the stack for the handler's privilege level.

If no stack switch occurs, the processor does the following when calling an interrupt or exception handler (see Figure 44):

1. Pushes the current contents of the EFLAGS, CS, and EIP registers (in that order) on the stack.
2. Pushes an error code (if appropriate) on the stack.
3. Loads the segment selector for the new code segment and the new instruction pointer (from the interrupt gate or trap gate) into the CS and EIP registers, respectively.
4. If the call is through an interrupt gate, clears the IF flag in the EFLAGS register.
5. Begins execution of the handler procedure at the new privilege level.

If a stack switch does occur, the processor does the following:

1. Temporarily saves (internally) the current contents of the SS, ESP, EFLAGS, CS, and EIP registers.
2. Loads the segment selector and stack pointer for the new stack (that is, the stack for the privilege level being called) from the TSS into the SS and ESP registers and switches to the new stack.
3. Pushes the temporarily saved SS, ESP, EFLAGS, CS, and EIP values for the interrupted procedure's stack onto the new stack.
4. Pushes an error code on the new stack (if appropriate).
5. Loads the segment selector for the new code segment and the new instruction pointer (from the interrupt gate or trap gate) into the CS and EIP registers, respectively.
6. If the call is through an interrupt gate, clears the IF flag in the EFLAGS register.
7. Begins execution of the handler procedure at the new privilege level.

A return from an interrupt or exception handler is initiated with the IRET instruction. The IRET instruction is similar to the far RET instruction, except that it also restores the contents of the EFLAGS register for the interrupted procedure:

When executing a return from an interrupt or exception handler from the same privilege level as the interrupted procedure, the processor performs these actions:

1. Restores the CS and EIP registers to their values prior to the interrupt or exception.
2. Restores the EFLAGS register.
3. Increments the stack pointer appropriately.
4. Resumes execution of the interrupted procedure.

When executing a return from an interrupt or exception handler from a different privilege level than the interrupted procedure, the processor performs these actions:

1. Performs a privilege check.
2. Restores the CS and EIP registers to their values prior to the interrupt or exception.
3. Restores the EFLAGS register.

4. Restores the SS and ESP registers to their values prior to the interrupt or exception, resulting in a stack switch back to the stack of the interrupted procedure.
5. Resumes execution of the interrupted procedure.

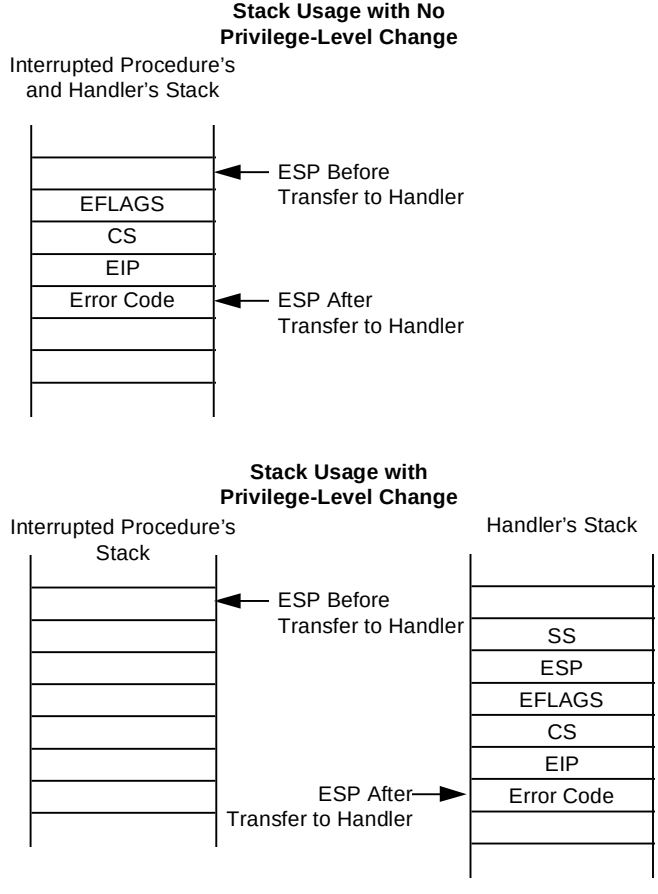


Figure 4-4. Stack Usage on Transfers to Interrupt and Exception Handling Routines

14. Performance-Monitoring Counter Issues

The following table replaces Table B-1 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide,* and Table A-1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. The only changes to this new table are enhanced descriptions of the events counted.

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
Data Cache Unit (DCU)	43H	DATA_ MEM_ REFS	00H	All loads from any memory type. All stores to any memory type. Each part of a split is counted separately. The internal logic counts not only external memory loads and stores, but also internal retries. Note: 80-bit floating-point accesses are double counted, since they are decomposed into a 16-bit exponent load and a 64-bit mantissa load. Memory accesses are only counted when they are actually performed. E.g., a load that gets squashed because a previous cache miss is outstanding to the same address, and which finally gets performed, is only counted once. Does not include I/O accesses, or other nonmemory accesses.	
	45H	DCU_LINES _IN	00H	Total lines allocated in the DCU.	
	46H	DCU_M_ LINES_IN	00H	Number of M state lines allocated in the DCU.	
	47H	DCU_M_ LINES_ OUT	00H	Number of M state lines evicted from the DCU. This includes evictions via snoop HITM, intervention or replacement.	
	48H	DCU_ MISS_ OUT-STAND-ING	00H	Weighted number of cycles while a DCU miss is outstanding, incremented by the number of outstanding cache misses at any particular time. Cacheable read requests only are	An access that also misses the L2 is short-changed by 2 cycles. (i.e., if count is N cycles, should be N+2 cycles.) Subsequent loads

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				considered. Uncacheable requests are excluded. Read-for-ownerships are counted as well as line fills, invalidates, and stores.	to the same cache line will not result in any additional counts. Count value not precise, but still useful.
Instr-uction Fetch Unit (IFU)	80H	IFU_ IFETCH	00H	Number of instruction fetches, both cacheable and noncacheable. Including UC fetches.	
	81H	IFU_ IFETCH_MISS	00H	Number of instruction fetch misses. All instruction fetches that do not hit the IFU, i.e., that produce memory requests. Includes UC accesses.	
	85H	ITLB_ MISS	00H	Number of ITLB misses.	
	86H	IFU_ MEM_ STALL	00H	Number of cycles instruction fetch is stalled, for any reason. Includes IFU cache misses, ITLB misses, ITLB faults and other minor stalls.	
	87H	ILD_ STALL	00H	Number of cycles that the instruction length decoder is stalled.	
L2 Cache[1]	28H	L2_ IFETCH	MESI 0FH	Number of L2 instruction fetches. This event indicates that a normal instruction fetch was received by the L2. The count includes only L2 cacheable instruction fetches; it does not include UC instruction fetches. It does not include ITLB miss accesses.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	29H	L2_LD	MESI 0FH	Number of L2 data loads. This event indicates that a normal, unlocked, load memory access was received by the L2. It includes only L2 cacheable memory accesses; it does not include I/O accesses, other nonmemory accesses, or memory accesses such as UC/WT memory accesses. It does include L2 cacheable TLB miss memory accesses.	
	2AH	L2_ST	MESI 0FH	Number of L2 data stores. This event indicates that a normal, unlocked, store memory access was received by the L2. Specifically, it indicates that the DCU sent a read-for-ownership request to the L2. It also includes Invalid to Modified requests sent by the DCU to the L2. It includes only L2 cacheable store memory accesses; it does not include I/O accesses, other nonmemory accesses, or memory accesses like UC/WT stores. It includes TLB miss memory accesses.	
	24H	L2_LINES _IN	00H	Number of lines allocated in the L2.	
	26H	L2_LINES_O UT	00H	Number of lines removed from the L2 for any reason.	
	25H	L2_M_ LINES_ INM	00H	Number of modified lines allocated in the L2.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	27H	L2_M_ LINES_ OUTM	00H	Number of modified lines removed from the L2 for any reason.	
	2EH	L2_ RQSTS	MESI 0FH	Total number of L2 requests.	
	21H	L2_ADS	00H	Number of L2 address strobes.	
	22H	L2_DBUS_B USY	00H	Number of cycles during which the L2 cache data bus was busy.	
	23H	L2_DBUS_B USY_ RD	00H	Number of cycles during which the data bus was busy transferring read data from L2 to the processor.	
Exter-nal Bus Logic (EBL)[2]	62H	BUS_ DRDY_ CLOCKS	00H (Self) 20H (Any)	Number of clocks during which DRDY# is asserted. Essentially, utilization of the external system data bus.	Unit Mask = 00H counts bus clocks when the processor is driving DRDY#. Unit Mask = 20H counts in processor clocks when any agent is driving DRDY#.
	63H	BUS_ LOCK_ CLOCKS	00H (Self) 20H (Any)	Number of clocks during which LOCK# is asserted on the external system bus.	Always counts in processor clocks.
	60H	BUS_REQ_ OUT-STAND-ING	00H (Self)	Number of bus requests outstanding. This counter is incremented by the number of cacheable read bus requests outstanding in any given cycle.	Counts only DCU full-line cacheable reads, not RFOs, writes, instruction fetches, or anything else. Counts “waiting for bus to complete” (last data chunk received).
	65H	BUS_ TRAN_ BRD	00H (Self) 20H (Any)	Number of burst read transactions.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	66H	BUS_ TRAN_ RFO	00H (Self) 20H (Any)	Number of completed read for ownership transactions.	
	67H	BUS_ TRANS_WB	00H (Self) 20H (Any)	Number of completed write back transactions.	
	68H	BUS_ TRAN_ IFETCH	00H (Self) 20H (Any)	Number of completed instruction fetch transactions.	
	69H	BUS_ TRAN_ INVAL	00H (Self) 20H (Any)	Number of completed invalidate transactions.	
	6AH	BUS_ TRAN_ PWR	00H (Self) 20H (Any)	Number of completed partial write transactions.	
	6BH	BUS_ TRANS_P	00H (Self) 20H (Any)	Number of completed partial transactions.	
	6CH	BUS_ TRANS_ IO	00H (Self) 20H (Any)	Number of completed I/O transactions.	
	6DH	BUS_ TRAN_ DEF	00H (Self) 20H (Any)	Number of completed deferred transactions.	
	6EH	BUS_ TRAN_ BURST	00H (Self) 20H (Any)	Number of completed burst transactions.	
	70H	BUS_ TRAN_ ANY	00H (Self) 20H (Any)	Number of all completed bus transactions. Address bus utilization can be calculated knowing the minimum address bus occupancy. Includes special cycles, etc.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	6FH	BUS_ TRAN_ MEM	00H (Self) 20H (Any)	Number of completed memory transactions.	
	64H	BUS_ DATA_ RCV	00H (Self)	Number of bus clock cycles during which this processor is receiving data.	
	61H	BUS_BNR_ DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the BNR# pin.	
	7AH	BUS_HIT_ DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HIT# pin.	Includes cycles due to snoop stalls. The event counts correctly, but the BPM*i* pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers). If the core clock to bus clock ratio is 2:1 or 3:1, and a PC bit is set, the BPM*i* pins will be asserted for a single clock when the counters overflow. If the PC bit is clear, the processor toggles the BPM*i* pins when the counter overflows. If the clock ratio is not 2:1 or 3:1, the BPM*i* pins will not function for these performance-monitoring counter events.

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
	7BH	BUS_ HITM_ DRV	00H (Self)	Number of bus clock cycles during which this processor is driving the HITM# pin.	Includes cycles due to snoop stalls. The event counts correctly, but the BPM*i* pins function as follows based on the setting of the PC bits (bit 19 in the PerfEvtSel0 and PerfEvtSel1 registers). If the core clock to bus clock ratio is 2:1 or 3:1, and a PC bit is set, the BPM*i* pins will be asserted for a single clock when the counters overflow. If the PC bit is clear, the processor toggles the BPM*i* pins when the counter overflows. If the clock ratio is not 2:1 or 3:1, the BPM*i* pins will not function for these performance-monitoring counter events.
	7EH	BUS_ SNOOP_ STALL	00H (Self)	Number of clock cycles during which the bus is snoop stalled.	
Floating Point Unit	C1H	FLOPS	00H	Number of computational floating-point operations retired. Excludes floating point computational operations that cause traps or assists. Includes floating point computational operations executed by the assist handler. Includes internal sub-operations of complex	Counter 0 only

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				floating point instructions like transcendentals. Excludes floating point loads and stores.	
	10H	FP_COMP_ OPS_ EXE	00H	Number of computational floating-point operations executed. The number of FADD, FSUB, FCOM, FMULs, integer MULs and IMULs, FDIVs, FPREMs, FSQRTS, integer DIVs and IDIVs. Note not the number of cycles but, the number of operations. This event does not distinguish an FADD used in the middle of a transcendental flow from a separate FADD instruction.	Counter 0 only
	11H	FP_ ASSIST	00H	Number of floating-point exception cases handled by microcode.	Counter 1 only. This event includes counts due to speculative execution.
	12H	MUL	00H	Number of multiplies. Note: includes integer and well FP multiplies and is speculative.	Counter 1 only
	13H	DIV	00H	Number of divides. Note: includes integer and FP multiplies and is speculative.	Counter 1 only
	14H	CYCLES_DI V_ BUSY	00H	Number of cycles that the divider is busy, and cannot accept new divides. Note: includes integer and FP divides, FPREM, FPSQRT, etc., and is speculative.	Counter 0 only
Mem-ory Order-ing	03H	LD_ BLOCKS	00H	Number of store buffer blocks. Includes counts caused by preceding stores whose addresses are	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
				unknown, preceding stores whose addresses are known to conflict, but whose data is unknown and preceding stores that conflicts with the load, but which incompletely overlap the load.	
	04H	SB_DRAINS	00H	Number of store buffer drain cycles. Incremented during every cycle the store buffer is draining. Draining is caused by serializing operations like CPUID, synchronizing operations like XCHG, Interrupt acknowledgment as well as other conditions such as cache flushing.	
	05H	MIS-ALIGN_ MEM_ REF	00H	Number of misaligned data memory references. Incremented by 1 every cycle during which either the PPro load or store pipeline dispatches a misaligned uop. Counting is performed if its the first half or second half, or if it is blocked, squashed or misses. Note in this context misaligned means crossing a 64 bit boundary.	It should be noted that MISALIGN_MEM_ REF is only an approximation, to the true number of misaligned memory references. The value returned is roughly proportional to the number of misaligned memory accesses, i.e., the size of the problem.
Instr-uction Decod-ing and Retire-ment	C0H	INST_ RETIRED	00H	Number of instructions retired.	A hardware interrupt received during/after the last iteration of the REP STOS flow causes the counter to undercount by 1 instruction.
	C2H	UOPS_	00H	Number of UOPs	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
		RETIRED		retired.	
	D0H	INST_ DECOD-ER	00H	Number of instructions decoded.	
Inter-rupts	C8H	HW_INT_RX	00H	Number of hardware interrupts received.	
	C6H	CYCLES_ INT_ MASKED	00H	Number of processor cycles for which interrupts are disabled.	
	C7H	CYCLES_ INT_ PENDING_A ND_ MASKED	00H	Number of processor cycles for which interrupts are disabled and interrupts are pending.	
Bran-ches	C4H	BR_INST_R ETIRED	00H	Number of branch instructions retired.	
	C5H	BR_MISS_P RED_ RETIRED	00H	Number of mispredicted branches retired.	
	C9H	BR_ TAKEN_ RETIRED	00H	Number of taken branches retired.	
	CAH	BR_MISS_P RED_ TAKEN_ RET	00H	Number of taken mispredictions branches retired.	
	E0H	BR_INST_ DECOD-ED	00H	Number of branch instructions decoded.	
	E2H	BTB_ MISSES	00H	Number of branches that for which the BTB did not produce a prediction	
	E4H	BR_ BOGUS	00H	Number of bogus branches.	
	E6H	BA-CLEARS	00H	Number of time BACLEAR is asserted. This is the number of times that a static branch prediction was made, where the branch decoder decided to make a branch prediction because the BTB did not.	

Unit	Event Number	Mnemonic Event Name	Unit Mask	Description	Comments
Stalls	A2H	RE-SOURCE_STALLS	00H	Incremented by one during every cycle that there is a resource related stall. Includes register renaming buffer entries, memory buffer entries. Does not include stalls due to bus queue full, too many cache misses, etc. In addition to resource related stalls, this event counts some other events. Includes stalls arising during branch misprediction recovery, e.g., if retirement of the mispredicted branch is delayed and stalls arising while store buffer is draining from synchronizing operations.	
	D2H	PARTIAL_RAT_ STALLS	00H	Number of cycles or events for partial stalls. Note Includes flag partial stalls.	
Segment Register Loads	06H	SEG-MENT_REG_ LOADS	00H	Number of segment register loads	
Clocks	79H	CPU_CLK_UN-HALTED	00H	Number of cycles during which the processor is not halted.	

NOTES:

1. Several L2 cache events, where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. The lower 4 bits of the Unit Mask field are used in conjunction with L2 events to indicate the cache state or cache states involved. The Pentium® Pro processor identifies cache states using the "MESI" protocol and consequently each bit in the Unit Mask field represents one of the four states: UMSK[3] = M (8H) state, UMSK[2] = E (4H) state, UMSK[1] = S (2H) state, and UMSK[0] = I (1H) state. UMSK[3:0] = MESI (FH) should be used to collect data for all states; UMSK = 0H, for the applicable events, will result in nothing being counted.
2. All of the external bus logic (EBL) events, except where noted, can be further qualified using the Unit Mask (UMSK) field in the PerfEvtSel0 and PerfEvtSel1 registers. Bit 5 of the UMSK field is used in conjunction with the EBL events to indicate whether the processor should count transactions that are self generated (UMSK[5] = 0) or transactions that result from any processor on the bus (UMSK[5] = 1).

15. Clock Jitter Measurement Clarification

The following text will be added to footnote 3 of Table 11-9, "Bus Clock A.C. Specifications," in the *Pentium® Pro Family Developer's Manual, Volume 1*.

"To ensure a 1:1 relationship between the amplitude of the input jitter and the internal and external clocks, the jitter frequency spectrum should not have any power spectrum peaking between 100kHz and 10 MHz. A spectrum analyzer can display the frequency spectrum of the clock driver in your system."

16. Exception Handler Error Code Bit Clarification

Section 5.10 on page 5-15 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual*, describes the bit definitions for the error code pushed onto the stack of the exception handler. The explanation of the EXT bit 0 will be changed to read as follows: External event (bit 0). When set, indicates that an event external to the program caused the exception*, such as a hardware interrup*t.

17. Clock Frequencies and Ratios and OverDrive® Processors

To clarify the *Pentium® Pro Family Developer's Manual, Volume 1: Specification*s, Section 9.2, "Clock Frequencies and Ratios," on page 9-9, the following Section 9.2.2 will be added:

9.2.2 Clock Frequencies and Ratios and OverDrive® Processors

The register values of the clock frequency ratio specified in the Power-on Configuration Register do not reflect a frequency multiplier when an OverDrive processor is installed. Rather, these register values reflect what has been sampled on the pins as specified in Table 9-4, "Bus Frequency to Core Frequency Ratio Configuration." Disregard the register values for the clock frequency when an OverDrive processor is installed.

18. BERR# Pin Description Clarification

The *Pentium® Pro Family Developer's Manual, Volume 1: Specifications,* Appendix A, "Signals Reference," shows in Section A.1.10, on page A-6, the BERR# pin description. The last paragraph is unclear in distinguishing the behavior of the Pentium Pro processor and a central agent. The paragraph below replaces the last paragraph in the BERR# description to help clarify each agent's role.

"BERR# sampling conditions are also defined by the system configuration. Configuration options enable the BERR# receiver to be enabled or disabled. When a central agent samples an active BERR# signal, it can forward the BERR# as an NMI or BINIT# to one of the processors. The Pentium Pro processor does not support BERR# sampling. (always disabled)"

19. IERR# Pin Description Clarification

The *Pentium® Pro Family Developer's Manual, Volume 1: Specifications,* Appendix A, "Signals Reference," shows in Section A.1.31, the IERR# pin description. The first paragraph could be interpreted such that IERR# deassertion can be accomplished via software. The paragraph below replaces the first paragraph in the description to make this clearer:

"The IERR# signal is the Error group Internal Error signal. A Pentium Pro processor asserts IERR# when it observes an internal error. It keeps IERR# asserted until it is turned off as part of the Machine Check Error handler, or with RESET# or BINIT# assertion. An NMI handler in software can indirectly accomplish the deassertion of IERR# via RESET# or BINIT# assertion."

20. Propagation of Page Table Entry Changes to Multiple Processors

The *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual,* describes techniques for Multiple Processor Management in Chapter 7. The following new section, which addresses TLB management in MP systems will be inserted for clarity, between Sections 7.2 and 7.3.

7.3 Propagation of Page Table Entry Changes to Multiple Processors

In a multiprocessor system, when one processor changes a page table entry or mapping, the changes must also be propagated to all of the other processors. This process is also known as "TLB Shootdown." Propagation may be done by memory based semaphores and/or interprocessor interrupts between processors. One naive but algorithmically correct TLB shootdown sequence for the Intel Architecture is:

1. Begin barrier: Stop all processors. Cause all but one to HALT or stop in a spinloop.
2. Let the active processor change the PTE(s).
3. Let all processors invalidate the PTE(s) modified in their TLBs.
4. End barrier: Resume all processors.

Alternate, performance-optimized, TLB shootdown algorithms may be developed, however, care must be taken by the developers to ensure that:

1. The differing TLB mappings are not actually used on different processors during the update process.

OR

2. The operating system is prepared to deal with the case where processor(s) are using the stale mapping during the update process.

21. APIC Bus Status Cycles Interpretation

The *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Manual,* in Section 7.4.13.2, "APIC Bus Status Cycles," shows Table 7-6, "APIC Bus Status Cycles Interpretation," on page 7-35. The table shown below contains more exhaustive information:

Delivery Mode	A Status	A1 Status	A2 Status	Update ArbID and Cycle#	Message Length	Retry
EOI	00: CS_OK	10: Accept	XX:	Yes, 13	14 cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 13	14 cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	14 cycle	Yes
	11: CS_Error	XX:	XX:	No	14 cycle	Yes
	10: Error	XX:	XX:	No	14 cycle	Yes
	01: Error	XX:	XX:	No	14 cycle	Yes
Fixed	00: CS_OK	10: Accept	XX:	Yes, 20	21 cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 cycle	Yes
	11: CS_Error	XX:	XX:	No	21 cycle	Yes
	10: Error	XX:	XX:	No	21 cycle	Yes
	01: Error	XX:	XX:	No	21 cycle	Yes
NMI, SMI, INIT, StartUp, ExtINT.	00: CS_OK	10: Accept	XX:	Yes, 20	21 cycle	No
	00: CS_OK	11: Retry	XX:	Yes, 20	21 cycle	Yes
	00: CS_OK	0X: Accept Error	XX:	No	21 cycle	Yes
	11: CS_Error	XX:	XX:	No	21 cycle	Yes
	10: Error	XX:	XX:	No	21 cycle	Yes
	01: Error	XX:	XX:	No	21 cycle	Yes
LOWEST	00: CS_OK, NoFocus	11: Do Lowest	10: Accept	Yes, 20	34 cycle	No
	00: CS_OK, NoFocus	11: Do Lowest	11: Error	Yes, 20	34 cycle	Yes
	00: CS_OK, NoFocus	11: Do Lowest	0X: Error	Yes, 20	34 cycle	Yes
	00: CS_OK, NoFocus	10: End & Retry	XX:	Yes, 20	34 cycle	Yes
	00: CS_OK, NoFocus	0X: Error	XX:	No	34 cycle	Yes
	10: CS_OK, Focus	XX:	XX:	Yes, 20	21 cycle	No
	11: CS_Error	XX:	XX:	No	21 cycle	Yes
	01: Error	XX:	XX:	No	21 cycle	Yes

22. Multiple Processor Protocol and Restrictions

Section 7.5.2 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, contain inconsistencies which will be clarified as follows:

7.5.2 Protocol Requirements and Restrictions

The MP protocol imposes the following requirements and restrictions on the system:

- An APIC clock (APICLK) must be provided on all systems based on P6 family processors (excluding Mobile processors and modules).
- All interrupt mechanisms must be disabled for the duration of the MP protocol algorithm including the window of time between the assertion of INIT# or receipt of an INIT IPI by the application processors and the receipt of a STARTUP IPI by the application processors. That is, requests generated by interrupting devices must not be seen by the local APIC unit (on board the processor) until the completion of the algorithm. Failure to disable the interrupt mechanisms may result in processor shutdown.
- The MP protocol should be initiated only after a hardware reset. After completion of the protocol algorithm, a flag is set in the APIC base MSR of the BSP (APIC_BASE.BSP) to indicate that it is the BSP. This flag is cleared for all other processors. If a processor or the system is subject to an INIT sequence (either through the INIT# pin or an INIT IPI), then the MP protocol is not re-executed. Instead, each processor examines its BSP flag to determine whether the processor should boot or wait for a STARTUP IPI.

23. NMI Handling While in SMM

Section 9.7, "NMI Handling While in SMM," in the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, will be clarified as follows:

9.7 NMI Handling While in SMM

NMI interrupts are blocked upon entry to the SMI handler. If an NMI request occurs during the SMI handler, it is latched and serviced after the processor exits SMM. Only one NMI request will be latched during the SMI handler. If an NMI request is pending when the processor executes the RSM instruction, the NMI is serviced before the next instruction of the interrupted code sequence.

Although the NMI requests are blocked when the CPU enters SMM, they may be enabled through software by executing an IRET/IRETD instruction. If the SMM handler requires the use of NMI interrupts, it should invoke a dummy interrupt service routine for the purpose of executing an IRET/IRETD instruction. Once an IRET/IRETD instruction is executed, NMI interrupt requests are serviced in the same "real mode" manner in which they are handled outside of SMM.

24. Critical Sequence of Events During a Page Fault Exception

Section 3.6.4 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, and the *Intel Architecture Software Writers Developers Manual, Volume 3: System Programming Guide*, will be clarified as follows:

If the processor generates a page-fault exception, the operating system must carry out the following operations in this order:

1. Copy the page from disk storage into physical memory if needed.
2. Load the page address into the page-table or page-directory entry and set its *present* flag. Other bits, such as the dirty and accessed bits, may also be set at this time.

3. Invalidate the current page table entry in the TLB (see Section 3.7, “Translation Lookaside Buffers (TLBs),” for a discussion of TLBs and how to invalidate them).
4. Return from the page fault handler to restart the interrupted program or task.

25. POP[ESP] with 16-bit Stack Size

In the *Pentium® Pro Family Developer’s Manual, Volume 2: Programmer’s Reference Manual*, in the section regarding “POP–Pop a Value from the Stack,” the following note:

“If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register.”

is incomplete, and should read as follows:

“If the ESP register is used as a base register for addressing a destination operand in memory, the POP instruction computes the effective address of the operand after it increments the ESP register. For the case of a 16-bit stack where ESP wraps to 0h as a result of the POP instruction, the resulting location of the memory write is processor family specific.”

In Section 15.12.1 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer’s Guide*, add a new section:

A POP-to-memory instruction, which uses the stack pointer (ESP) as a base register.

For a POP-to-memory instruction that meets the following conditions:

1. The stack segment size is 16-bit
2. Any 32-bit addressing form with the SIB byte specifying ESP as the base register
3. The initial stack pointer is FFFCh(32-bit operand) or FFFEh (16-bit operand) and will wrap around to 0h as a result of the POP operation

The result of the memory write is processor family specific. For example, in Pentium II and Pentium Pro processors the result of the memory write is to SS:0h plus any scaled index and displacement. In Pentium and i486™ processors, the result of the memory write may be either a stack fault (real mode or protected mode with stack segment size of 64 Kbyte), or write to SS:10000h plus any scaled index and displacement (protected mode and stack segment size exceeds 64 Kbyte).

26. Paging Must Be Enabled Before Enabling the Page Global Bit

In Section 2.5 of the *Intel Architecture Software Developer’s Manual, Volume 3: System Programming Guide,* and in Section 2.5 of the *Pentium® Pro Family Developer’s Manual, Volume 3: Operating System Writer’s Manual*, the following line should be added to the text describing the Page Global Enable bit (PGE).

“In addition, the bit must not be enabled before paging is enabled via CR0.PG. Program correctness may be affected by reversing this sequence, and processor performance will be impacted.”

27. SMIACK Transaction Generated on Exiting from SMM

In the *Pentium® Pro Family Developer’s Manual, Volume 3: Operating Systems Writer’s Manual*, Section 9.3.1.1 and in the *Intel Architecture Software Developer’s Manual, Volume 3: System Programming Guide*, Section 11.3.1.1, the paragraph stating,

"The RSM instruction restores the processor's context by loading the state save image from SMRAM back into the processor's registers. It then returns program control back to the interrupted program."

should state the following:

"The RSM instruction restores the processor's context by loading the state save image from SMRAM back into the processor's registers. The processor then issues an SMIACK transaction on the system bus and returns program control back to the interrupted program."

28. Software Initialization Requirements for FRC Mode

In the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide*, Section 8.4, and in the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Section 8.4, the following paragraph should be added to the end of each of the sections:

"Systems configured to implement FRC mode must write all of the processors' internal MSRs to deterministic values before performing either a read or read-modify-write operation using these registers. The following is a list of MSRs that are not initialized by the processors' reset sequences.

1. All fixed and variable MTRRs,
2. All Machine Check Architecture (MCA) status registers,
3. Microcode Update signature register, and
4. All L2 Cache initialization MSRs."

29. Switching to Protected Mode While in SMM

Should the System Management Mode (SMM) code developer require a transition to protected mode while in SMM, a change is required to the sequence of events used to switch to protected mode as documented in Section 8.8.1 of the *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*.

Items 3 and 4 of this section state:

3. Execute a MOV CR0 instruction that sets the PE flag (and optionally the PG flag) in control register CR0.
4. Immediately following the MOV CR0 instruction, execute a far JMP or far CALL instruction. (This operation is typically a far jump or call to the next instruction in the instruction stream.)

Random failures can occur if other instructions exist between steps 3 and 4, and failures will be readily seen in some situations such as when instructions that reference memory are inserted between steps 3 and 4 above while in System Management Mode.

SPECIFICATION CHANGES

The Specification Changes listed in this section apply to the *Pentium® Pro Family Developer's Manual*, *Volume 1* (Order Number 242690), or the *Pentium® Pro Processor at 150 MHz, 166 MHz, 180 MHz, and 200 MHz* datasheet (Order Number 242769), or the *Pentium® Pro Processor with 1 MB L2 Cache at 200 MHz* datasheet (Order Number 243570-001). All Specification Changes will be incorporated into future versions of the appropriate document(s).

1. Mixing Steppings in MP Systems

Though Intel recommends using identical steppings of processor silicon in multiprocessor systems whenever possible (as this is the only configuration which receives full validation across all of Intel's testing), Intel supports mixing processor steppings, and does not actively prevent various steppings of the Pentium Pro processor from working together in MP systems. However, since Intel cannot validate every possible combination of devices, each new stepping of a device is fully validated only against the latest steppings of other processors and chipset components.

The following list and matrix explain the known issues with mixing steppings:

- While Intel has done nothing to prevent different frequency Pentium® Pro processors within a system from working together, there may be uncharacterized errata which exist in such configurations. In mixed stepping systems, all processors must be run at an identical frequency (i.e., the *highest* frequency acceptable to all components).
- The workarounds for various errata must take all processors into account.
- Errata for all processor steppings present in a system will affect that system, unless worked around.
- The following notes apply only to specific combinations:
  1. The 150-MHz and the 166-, 180-, and 200-MHz parts vary in their GTL+ T_{CO_MIN} and T_{HOLD} timings by 250 ps. This may affect some system designs. Also, a customer may wish to use a higher frequency replacement for a 150-MHz processor in a 150-MHz system; this should also be considered.
  2. OverDrive® processor upgrades for the Pentium Pro processor will only be supported in one and two processor systems, and are only supported in same stepping configurations.
  3. Errata 39 and 41 may be problematic for 3- or 4-way MP systems which use processor steppings previous to the sA1 stepping.

In the following table, "NI" implies that there are currently no known issues associated with mixing these steppings. An "X" implies that these steppings should not be used together in a system. A number indicates a known issue, and refers to the numbered list above. While there are no currently known issues associated with mixing cache sizes, Intel does not recommend or validate mixing processors with other cache sizes.

Pentium® Pro Processor Stepping	150-MHz B0	150-MHz C0	166-, 180-, 200-MHz sA0	166-, 180-, 200-MHz sA1	166-, 180-, 200-MHz sB1	OverDrive® Processor
150-MHz B0	3	3	1, 3	1, 3	1, 3	X
150-MHz C0	3	3	1, 3	1, 3	1, 3	X
166-, 180-, 200-MHz sA0	1, 3	1, 3	3	3	3	X
166-, 180-, 200-MHz sA1	1, 3	1, 3	3	NI	NI	X
166-, 180-, 200-MHz sB1	1, 3	1, 3	3	NI	NI	X
OverDrive® Processor	X	X	X	X	X	2

2. Change in PICD[1:0] Hold Time and Valid Delay

The minimum value of the hold time for the PICD[1:0] signals, T28, will be changed from 2 ns to 2.5 ns. The minimum value of the valid delay for these signals, T29, will be changed from 2.1 ns to 2.0 ns. These values change those documented in Table 11-15 of the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications,* and Table 15 of the *Pentium® Pro Processor with 1 MB L2 Cache at 200 MHz* datasheet.

3. Undershoot Specification for Some GTL+ Signals

The *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*, Section 11.15, will include the following undershoot specification, to simplify signal integrity issues on signals that can be driven by more than one device simultaneously:

The Pentium Pro processor bus signals AERR#, BERR#, BINIT#, BNR#, HIT#, and HITM# (only) are capable of sinking an 85 mA current pulse at 2.4% average time duty cycle. This is equivalent to -1.7 V applied to a 20Ω source in series with the device pin for 8 ns at 66 MHz with a utilization of 5%.

Note: This test covers the AC operating conditions only.

4. OverDrive® VRM Motherboard Bulk Capacitance Specification

There is a limit to the amount of bulk capacitance which may be moved to the motherboard and still allow the system to be upgraded with an OverDrive® processor and OverDrive VRM. In the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*, Section 17.4.1.2, "OverDrive® VRM D.C. Specifications," the following entry will be added to Table 17-5 and in the *Pentium® Pro Processor with 1 MB L2 Cache at 200 MHz* datasheet, Section 8.4.12, "OverDrive® VRM D.C. Specifications," Table 34.

Symbol	Parameter	Min	Max	Unit	Notes
Cmb	Total bulk capacitance between Header 8 and Socket 8		9000	µF	

5. MTRR Precedences and Memory Type Overriding

Part 2 of Section 11.11.4.1 of the *Pentium® Pro Family Developer's Manual, Volume 3: Operating System Writer's Guide,* will change as follows:

a. If one variable memory range matches, the processor uses the memory type stored in the MTRRphysBase*n* register for that range.

b. If two or more variable memory ranges match and the memory types are identical, then that memory type is used.

c. If two or more variable memory ranges match and one of the memory types is UC, the UC memory type is used.

d. If two or more variable memory ranges match and the memory types are WT and WB, the WT memory type is used.

e. If two or more variable memory ranges match and the memory types are not combinations as specified in parts a. through d. above, the behavior of the processor is undefined.

6. V_{CC}S Pins Removed From Specification

The Pentium Pro family will not use the V_{CC}S pins. All references to these pins will be removed from the specification. These references currently appear in the *Pentium® Pro Family Developer's Manual, Volume 1: Specifications,* Sections 11.3, 11.4.1, 11.8.1, 11.12, 11.13, and 17.4. The references to V_{CC}S in Section 15.2, "Pinout," will be changed to Reserved. However, connection to 3.3 V is still allowed for these signals.

7. Locks Across Cache Line Boundary Disable Bit Added

As of the sB1 stepping of the Pentium Pro processor, setting bit 31 of the Model Specific Register (MSR) at address 33h to '1' will prevent LOCK# from being asserted when locked transactions which are split across a cache line boundary are issued from the processor. This bit is disabled by default and remains Reserved for all previous steppings of the Pentium Pro processor. In the default state ('0'), unaligned data issued in a locked sequence by the processor will have atomicity with the LOCK# signal asserted. When the bit is set, any transactions issued which split a cache line boundary will not have the LOCK# signal asserted, and no atomicity can be guaranteed between the reads and writes in the sequence. Locked sequences which do not split a cache line boundary will still follow the normal LOCK# protocol with this bit set.

8. Instruction Streaming Buffer Disable Bit Added

As of the sB1 stepping of the processor, bit 30 of the Model Specific Register (MSR) at address 33h can be set to '1' to disable Instruction Streaming Buffers. [Note that on the sB1 stepping of the Pentium Pro processor, this bit can be used to work around Errata 58 and 59.] By default, the bit is '0', and Instruction Streaming Buffers are enabled. The bit is Reserved for all previous steppings of the processor and should not be written. On steppings prior to the sB1, attempts to write to this bit will be silently ignored by the processor.

9. System Bus Timings Changes

In Table 11-13 of *Pentium® Pro Family Developer's Manual, Volume 1: Specifications*, the following changes should be made:

Table 13. [1, 2, 3]

T#	Parameter	Min	Max	Unit	Figure	Notes
T11:	3.3 V Output Valid Delay	1.00	8	ns	11-8	1
T12:	3.3 V Input Setup Time	5		ns	11-9	2, 3, 4, 5
T13:	3.3 V Input Hold Time	1.5		ns	11-9	
T14:	3.3 V Input Pulse Width, except PWRGOOD	2		BCLKs	11-8	Active and Inactive states
T14B:	LINT[1:0] Input Pulse With	6		BLCKs	11-8	7
T15:	PWRGOOD Inactive Pulse Width	10		BCLKs	11-8 11-13	6

NOTES:

1. Valid delay timing for these signals are specified into 150Ω to 3.3 V. See Figure 11-6 for a capacitive derating curve.
2. These signals may be driven asynchronously. However, to guarantee recognition on a specific clock, the setup and hold times with respect to BCLK must be met.
3. These signals must be driven synchronously in FRC mode.
4. A20#, IGNNE#, INIT# and FLUSH# can be asynchronous inputs, but to guarantee recognition of these signals following a synchronizing instruction such as an I/O write instruction, they must be valid with active RS[2:0]# signals of the corresponding synchronizing bus transaction.
5. INTR and NMI are only valid in APIC disable mode. LINT[1:0]# are only valid in APIC enabled mode.
6. When driven inactive, or after Power, V_{REF}, BCLK, and the ratio signals are stable.
7. This specification only applies when the APIC is enabled and the LINT1 or LINT0 pin is configured as an edge triggered interrupt with fixed delivery, otherwise specification T14 applies.

10. Minimum Storage Specification Change for Processors with 1 MB L2 Cache

The Minimum Storage Temperature and Minimum Case Temperature Under Bias specifications will change from –65 °C to –55 °C. These values change those documented in Table 3 of the *Pentium® Pro Processor with 1 MB L2 Cache at 200 MHz* datasheet.

11. WC Buffer Eviction Data Ordering

The *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide,* states in Section 9.3.1 that "a completely full WC [Write Combining] buffer will always be propagated as a single burst transaction with ascending data order." This statement is incorrect and should be changed to "a completely full WC buffer will always be propagated as a single burst transaction using any of the valid chunk orders."